Aula 00 - Prof. Felipe Mathias e Raphael Lacerda

BANESE (Técnico Bancário III - Suporte) Desenvolvimento de Software - 2025 (Pós-Edital)

Autor:

Felipe Mathias, Paolla Ramos

21 de Janeiro de 2025

Índice

1) Apresentação - Felipe Mathias 3
2) Spring Framework - Teoria 4
3) Spring Framework - Questões Comentadas 38
4) Spring Framework - Lista de Questões 49
5) Spring Boot - Teoria 56
6) Spring Boot - Questões Comentadas 73
7) Spring Boot - Lista de Questões 82
8) Spring Cloud - Teoria 88
9) Spring Cloud - Questões Comentadas 100
10) Spring Cloud - Lista de Questões 108
11) Map Struct - Teoria 113
12) Map Struct - Questões Comentadas 118
13) Map Struct - Lista de Questões 121
14) Swagger - Teoria 124
15) Swagger - Questões Comentadas 132
16) Swagger - Lista de Questões 141

APRESENTAÇÃO DA AULA

Olá, alunos! Bem-vindos a mais uma aula do curso de Tecnologia de Informação para concursos públicos, no Estratégia Concursos.

Me chamo Felipe Mathias e serei seu professor na aula de hoje. Sou um catarinense de 30 anos, programador *front end* (ex-programador, se preferirem haha) e atuo como professor de cursos de Tecnologia da Informação voltados a concursos há mais de um ano. Assim como você, também vivo a vida de concurseiro, aguardando minha nomeação como Auditor Fiscal da Secretaria de Fazenda de Minas Gerais (SEF-MG), onde figuro no cadastro de reserva. Atualmente, continuo, em paralelo, estudando para concursos aguardando o meu grande sonho – o cargo de Auditor Fiscal da SEF-SC, com especialidade em TI.

Minha aventura no mundo do ensino surgiu de uma vontade interna de atuar como professor – sempre amei explicar as coisas, além de ter certa facilidade em expressar conceitos mais complexos para pessoas que talvez não tenham tanta experiência na área.

Meu objetivo aqui é digerir assuntos, desde os mais simples aos mais complexos, para que qualquer aluno consiga os entender, seja um programador, operador de infraestrutura, ou simplesmente um leigo que resolveu adentrar no mundo dos concursos e se deparou com TI no seu edital.

Gostaria de pedir que **sempre** vejam as questões comentadas durante a aula. Elas trazem conteúdo essencial para o aprendizado, muitas vezes abordando alguns pontos que não foram abordados no conteúdo e são essenciais para a resolução de questões.

Caso tenha alguma dúvida, não tenha receio de entrar em contato comigo nas minhas redes sociais (especialmente no meu Instagram, que deixarei abaixo), ou no fórum de dúvidas que os responderei assim que possível.

Ah, posto bastante coisa interessante de TI direcionada para concursos lá, dá uma olhadinha que algumas coisas podem te interessar. Volta e meio acerto alguma questão de prova por lá ;)

@fe.fiscal

t.me/fefiscal

SPRING FRAMEWORK

Conceitos Gerais

O **Spring** pode ser definido como um **framework de desenvolvimento para aplicações Java**, que fornece soluções para diversos problemas comuns encontrados no desenvolvimento de software, como gerenciamento de dependências, configuração de transações, integração com bancos de dados e construção de APIs RESTful. Seu grande diferencial está na flexibilidade e na modularidade, que permitem que os desenvolvedores utilizem apenas os componentes de que precisam, sem a necessidade de carregar o framework inteiro.

O desenvolvimento em Java, especialmente em sistemas corporativos, pode se tornar complexo. Em cenários tradicionais, os desenvolvedores frequentemente lidavam com configurações longas e intrincadas, especialmente quando o assunto era integrar diferentes partes da aplicação, como bancos de dados e interfaces web. O Spring surgiu para simplificar esses desafios, oferecendo uma abordagem baseada em Inversão de Controle (IoC) e Injeção de Dependências (DI), que explicaremos em detalhes em capítulos posteriores.

Imagine, por exemplo, que uma aplicação precisa acessar um banco de dados. Antes do Spring, seria necessário configurar manualmente cada detalhe da conexão, escrever longos códigos repetitivos e gerenciar recursos, como abrir e fechar conexões. Com o Spring, todo esse processo é simplificado. A configuração pode ser centralizada e reaproveitada, o que reduz a probabilidade de erros e facilita a manutenção.

Então, de forma geral, podemos elencar alguns objetivos que o Spring busca atingir:

- **Facilitar o desenvolvimento:** Com menos código boilerplate (ou seja, código repetitivo), os desenvolvedores podem se concentrar nas regras de negócio.
- **Promover a modularidade:** O Spring é organizado em módulos, como Spring Core, Spring Data e Spring MVC. Isso significa que podemos usar apenas o que for necessário para o projeto.
- **Aumentar a testabilidade:** O uso de IoC e DI torna o código mais testável, facilitando a criação de testes de unidade e integração.
- **Promover boas práticas:** O framework incentiva o uso de design patterns e princípios sólidos de desenvolvimento, como o SOLID.

Vamos explorar o Framework!

Inversão de Controle e Injeção de Dependência

Antes de adentrarmos no Spring, eu preciso que você saiba os conceitos de **Inversão de Controle**, ou **Inversion of Control (IoC),** e sobre a **Injeção de Dependências**, ou **Dependency Injection (DI)** – já que o Spring é pautado nesses conceitos.

A **IoC** é um princípio de design de software que se refere à **delegação do controle do fluxo de execução ou da criação de objetos de um programa para um framework ou container**. No contexto do Spring Framework, IoC é a base fundamental para o gerenciamento de dependências entre os componentes de uma aplicação, sendo implementada através de técnicas como a DI.

Tradicionalmente, em sistemas desenvolvidos sem IoC, os objetos de uma aplicação são responsáveis por criar e gerenciar suas próprias dependências. Esse modelo gera forte acoplamento entre os componentes, dificultando a manutenção, o teste e a escalabilidade do software. Com IoC, essa responsabilidade é transferida para um container (no caso do Spring, o ApplicationContext ou *Bean*Factory), que instancia e gerencia os objetos e suas dependências, promovendo uma arquitetura mais modular e flexível.

O conceito de IoC pode ser explicado pela inversão da lógica tradicional de controle. Em vez de o código da aplicação controlar diretamente o fluxo ou as dependências, o controle é "invertido" e delegado ao framework. Vamos entender isso com um exemplo simples. Com isso, alcançamos alguns benefícios:

- **Desacoplamento:** Os componentes de uma aplicação não precisam saber como suas dependências são instanciadas ou gerenciadas, facilitando substituições e alterações.
- **Testabilidade:** Como as dependências podem ser facilmente injetadas, é possível usar mocks ou stubs em testes unitários.
- **Flexibilidade:** Alterações no comportamento da aplicação (como trocar uma implementação de dependência) podem ser feitas sem modificar o código principal.
- **Manutenção:** A IoC promove o uso de boas práticas de design, como a separação de responsabilidades, tornando o código mais legível e fácil de manter.

(FUNDATEC/PROCERGS/2023) Uma prática utilizada em projetos de desenvolvimento de software orientado a objetos é a inversão de controle. Uma das vantagens da inversão de controle é:

a) Facilitar os testes.
b) Aumentar o acoplamento.
c) Reduzir a portabilidade.
d) Simplificar os controles de segurança.
e) Padronizar as integrações com serviços externos.

Comentários:

Vamos analisar as alternativas.

a) Certo. De fato, a IoC reduz o acoplamento, permitindo o uso de mocks e facilitando testes unitários.
b) Errado. IoC reduz o acoplamento, promovendo independência entre componentes.
c) Errado. IoC aumenta a portabilidade ao facilitar a substituição de dependências.
d) Errado. IoC não tem relação direta com simplificação de segurança.

e) Errado. IoC não padroniza integrações, mas facilita a injeção de dependências.

Portanto, correta a letra A. *(Gabarito: Letra A)*

Já a **Injeção de Dependências (DI)** é um padrão de design de software que facilita o gerenciamento e a inversão do controle das dependências entre os componentes de uma aplicação. Esse mecanismo permite que o framework forneça (ou injete) as dependências necessárias para os objetos da aplicação, em vez de os próprios objetos serem responsáveis por criar ou localizar essas dependências.

Antes de entender como a injeção funciona, é importante compreender **o que é uma dependência**. No desenvolvimento de software, uma **dependência ocorre quando uma classe utiliza outra classe para realizar sua funcionalidade**. Por exemplo, imagine uma classe `PedidoService` que depende de uma classe `PagamentoService` para processar pagamentos. Sem DI, `PedidoService` teria que instanciar manualmente `PagamentoService`, o que cria um forte acoplamento entre elas. Com a DI, quem gerencia essa dependência é o container do Spring. Ele cria as instâncias necessárias, resolve as dependências e injeta os objetos prontos na classe `PedidoService`.

A Injeção de Dependências ocorre quando um objeto recebe suas dependências de uma **fonte externa**, em vez de criá-las internamente. No Spring, essa fonte é o container IoC, que é responsável por instanciar e fornecer os objetos necessários. A DI pode ser realizada de três formas principais:

1. **Injeção via Construtor**
2. **Injeção via Setter**
3. **Injeção via Campo**

(FUNDATEC/PROCERGS/2023) Qual é a diferença entre inversão de controle e injeção de dependências?

a) A inversão de controle é um princípio e a injeção de dependências é uma técnica.
b) A inversão de controle é uma técnica e a injeção de dependências é um princípio.
c) Não há diferença, os termos são sinônimos.
d) A inversão de controle é um conceito antigo e a injeção de dependências é um conceito moderno.
e) A injeção de dependências é uma técnica específica da linguagem Java.

Comentários:

A IoC e DI andam "lado a lado" – a IoC agindo como um princípio, um conceito, e a DI agindo como uma técnica que implementa a IoC. *(Gabarito: Letra A)*

No Spring, os contêineres que formam a base para o IoC e a DI são baseados em **dois pacotes**:

- **org.springframework.*beans*:** contém interfaces e classes para manipulação de Java *beans*
- **org.springframework.context:** se baseia no pacote *beans* para incluir suporte para origens de mensagens e para o padrão de design Observer, e a capacidade de objetos de aplicativo para obter recursos usando uma API consistente

Além disso, temos outros pacotes associados a outras funcionalidades do Framework:

- **org.springframework.orm:** oferece suporte à integração com frameworks de mapeamento objeto-relacional (ORM), como o Hibernate, fornecendo classes utilitárias, abstrações de transação e gerenciamento simplificado de sessões e entidades, facilitando a interação entre o mundo orientado a objetos e os bancos de dados relacionais.
- **org.springframework.jdbc:** fornece uma camada de abstração sobre a API JDBC, reduzindo a quantidade de código repetitivo ao gerenciar conexões, tratar exceções e executar consultas SQL. Com isso, torna-se mais simples e seguro acessar dados em bancos relacionais, além de facilitar transações e mapeamentos de resultados.
- **org.springframework.web:** concentra o suporte ao desenvolvimento de aplicações web, incluindo a implementação do padrão MVC (Model-View-Controller), o gerenciamento de requisições HTTP, a configuração de roteamento, a conversão de dados, a validação de entradas e o suporte a diversos formatos de resposta, facilitando a criação de aplicações web robustas e escaláveis.
- **org.springframework.core:** é o núcleo fundamental do framework, reunindo interfaces, classes utilitárias e funcionalidades básicas que dão suporte às outras partes do Spring. Ele inclui sistemas de reflexão aprimorada, gerenciamento de recursos, tratamento de exceções e outros utilitários que formam a base sobre a qual os demais módulos são construídos.
- **org.springframework.expression:** fornece a linguagem de expressões do Spring (SpEL), permitindo avaliar e manipular propriedades, objetos, coleções, arrays, métodos e funções em tempo de execução. Essa capacidade torna o código mais flexível e declarativo, facilitando configurações dinâmicas e personalizações sem alterar a lógica principal da aplicação.

(FCC/TT 23/2022) A base do contêiner Inversion of Control (IoC), também conhecido como Dependency Injection (DI), do Spring Framework, é formada pelos pacotes

a) org.springframework.*beans* e org.springframework.context
b) org.springframework.orm e org.springframework.jdbc
c) org.springframework.web e org.springframework.webmvc
d) org.springframework.core e org.springframework.expression
e) org.springframework.webmvc e org.springframework.websocket

Comentários:

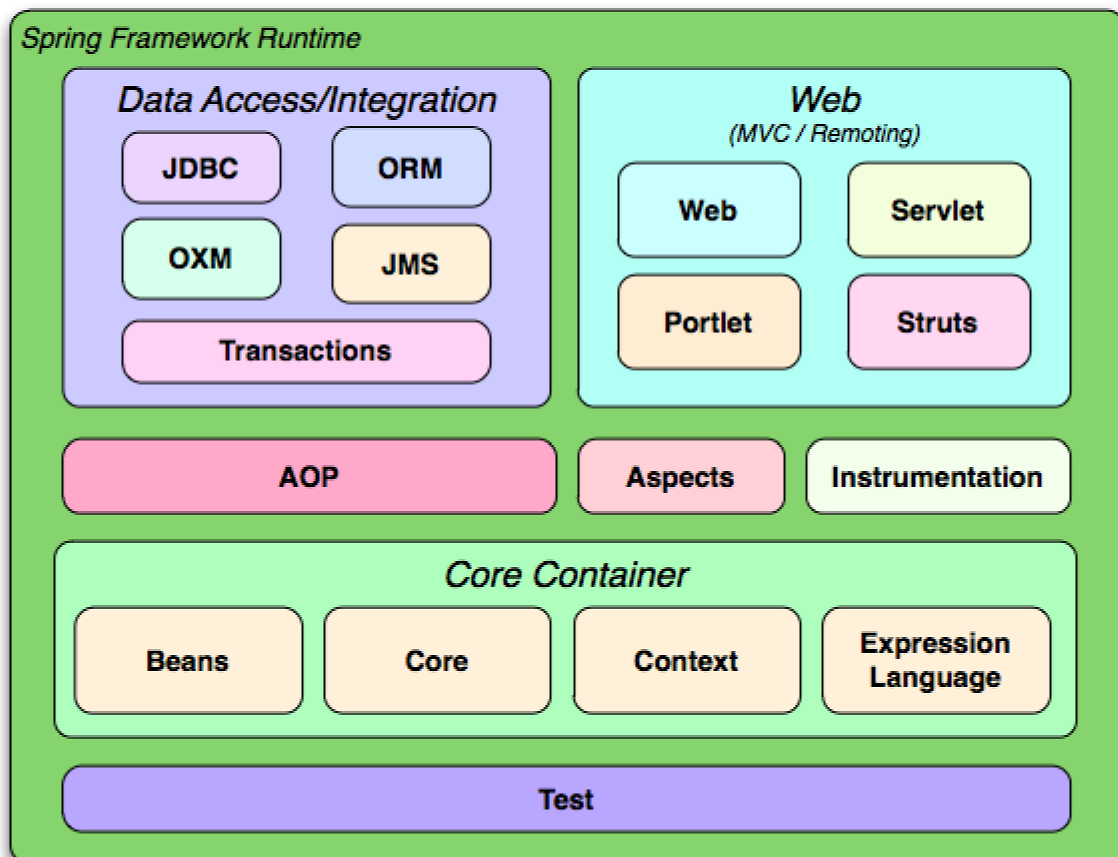
Como vimos, os pacotes responsáveis pelo IoC e DI são o ...*beans* e o ...context. *(Gabarito: Letra A)*

Arquitetura

O Spring Framework é projetado com uma **arquitetura modular**, o que significa que podemos usar apenas os componentes necessários para atender às demandas do nosso projeto. Essa abordagem permite o desenvolvimento de aplicações leves e flexíveis, evitando a sobrecarga de funcionalidades desnecessárias.

Essa arquitetura modular é dividida em diferentes camadas, que interagem entre si para formar as aplicações. Podemos citar as seguintes camadas:

- **Core Container**: A base de todo o framework, onde encontramos o suporte para IoC (Inversão de Controle).
- **Camada de Integração e Acesso a Dados**: Inclui suporte para JDBC, ORM e transações.
- **Camada Web**: Responsável por aplicações web e APIs REST.
- **Suporte à Programação Orientada a Aspectos (AOP)**: Permite separar preocupações transversais do código principal.
- **Testes e suporte a testes**: Ferramentas para facilitar a criação de testes automatizados



Fonte: docs.spring.io

Vamos explorar esses módulos, por camadas.

Core Container e *Beans*

O **Core Container** é a base de todo o ecossistema do Spring, sendo responsável por **gerenciar o ciclo de vida dos objetos** (chamados de ***beans***) de forma centralizada e independente das classes que os consomem. Ao entender como funciona o Core Container, podemos compreender melhor a essência do Spring, pois é nessa camada que ocorrem os processos fundamentais de Inversão de Controle (IoC) e Injeção de Dependências (DI).

Para entendermos o Core Container, precisamos pensar em um espaço onde as definições dos nossos objetos são registradas, e o próprio framework se encarrega de criá-los, gerenciá-los e disponibilizá-los para outras partes da aplicação. Esse espaço é controlado por um container de IoC, sendo o **`BeanFactory`** a interface raiz mais simples, e o **`ApplicationContext`**, uma extensão mais completa, a implementação mais utilizada na maioria das aplicações. O `ApplicationContext` oferece funcionalidades adicionais, como suporte a mensagens internacionais, carregamento de arquivos de configuração, integração com ambientes e, por fim, a capacidade de registrar *beans* de forma automática utilizando anotações, algo cada vez mais comum com a evolução do Spring.

Podemos iniciar o estudo pelo `BeanFactory`, que é a interface mínima capaz de instanciar e fornecer *beans*. Entretanto, a maioria dos projetos reais utiliza implementações do `ApplicationContext`, que herdam do `BeanFactory` e oferecem recursos mais abrangentes. O `ApplicationContext` pode ser configurado de diferentes maneiras, seja por meio de arquivos XML, classes Java com anotações ou até mesmo através de convenções baseadas no *classpath*. Com a evolução do framework e o surgimento do Spring Boot, a abordagem por anotações e convenções tornou-se a mais utilizada, tornando a necessidade de configurações XML cada vez menor.

> As anotações em Java são metadados inseridos diretamente no código-fonte que nos permitem transmitir informações adicionais ao compilador, às ferramentas de desenvolvimento e, principalmente, a frameworks e bibliotecas como o Spring. Elas não alteram a lógica de execução do programa em si, mas indicam comportamentos, regras de configuração ou instruções especiais que o ambiente de execução ou outras partes da aplicação devem considerar, tornando o código mais declarativo, legível e de fácil manutenção.

Para organizar o código, o **Spring utiliza o conceito de *beans***, que são **objetos gerenciados pelo container**. Podemos definir um *bean* como uma classe que desempenha um papel na lógica de negócio do nosso sistema e que, ao ser registrada no container, tem seu ciclo de vida controlado: o container cria a instância, injeta suas dependências, chama callbacks de inicialização, e, finalmente, pode destruir o *bean* quando não for mais necessário. Essa abordagem garante que não precisemos lidar manualmente com detalhes de criação, interligação e descarte de objetos, deixando a aplicação mais limpa e voltada à resolução do problema de negócio.

Veja um exemplo:

```java
@Component
public class ServicoNotificacao {
    public void enviar(String mensagem) {
        System.out.println("Notificação enviada: " + mensagem);
    }
}
```

```java
@Component
public class ServicoPedido {

    private final ServicoNotificacao servicoNotificacao;

    @Autowired
    public ServicoPedido(ServicoNotificacao servicoNotificacao) {
        this.servicoNotificacao = servicoNotificacao;
    }

    public void criarPedido() {
        // Lógica de criação do pedido
        servicoNotificacao.enviar("Pedido criado com sucesso!");
    }
}
```

Aqui, anotamos as classes com `@Component` para informar ao container que elas devem ser gerenciadas como *beans*. O `@Autowired` no construtor de `ServicoPedido` diz ao container para injetar automaticamente uma instância de `ServicoNotificacao`. Quando o aplicativo é iniciado, o Spring faz o scanning do pacote, encontra as classes anotadas, registra-as como *beans* e cuida da injeção de dependências. Essa abordagem baseada em convenções é mais declarativa e reduz a verbosidade do código de configuração.

Não se preocupe em saber as anotações ainda, teremos um capítulo só para isso.

Outro ponto importante do Core Container é a flexibilidade na resolução de recursos e a interação com o ambiente de execução. Podemos injetar valores a partir de arquivos externos, usar perfis (profiles) para carregar configurações específicas conforme o ambiente (desenvolvimento, teste, produção) e até mesmo fazer uso da Expression Language do Spring para definir propriedades dinamicamente. Todos esses recursos são gerenciados pelo container.

Além disso, o container suporta mecanismos de "lifecycle callbacks", o que significa que podemos executar ações específicas após a criação dos *beans* ou antes da sua destruição. Por exemplo, se precisarmos estabelecer uma conexão ou carregar dados específicos assim que um *bean* for criado, podemos implementar a interface `InitializingBean` ou usar a anotação `@PostConstruct` em métodos dedicados. Da mesma forma, se precisarmos fechar recursos externos (como conexões com sistemas remotos) antes que o *bean* seja removido, podemos utilizar `@PreDestroy`.

(COMPERVE/UFRN/2023) O Spring framework é uma ferramenta amplamente utilizada no desenvolvimento de aplicações Java Web. No ciclo de vida de um Spring *Bean*, é possível utilizar anotações em métodos que vão adicionar algum comportamento nos momentos de criação e na destruição desse *Bean*. As duas anotações utilizadas no spring para usar esses métodos customizados, no ciclo de vida dos *Beans*, são

a) @PostConstruct e @PreDestroy.
b) @PostConstruct e @BeforeDestroy.
c) @AfterConstruct e @PreDestroy.
d) @AfterConstruct e @BeforeDestroy.

Comentários:

Para incluir ações no momento da criação, usamos o @PostConstruct; já para incluir no momento de destriuição, usamos o @PreDestroy. *(Gabarito: Letra A)*

Ainda, sobre os *beans*, é possível definir diferentes configurações para seu escopo. Por padrão, **se nenhuma abordagem for especificada**, o **escopo do *bean* segue o padrão Singleton.** Mas temos outras possibilidades, veja:

- **Singleton (Padrão):** É o escopo padrão. Quando nenhum escopo é especificado, o Spring cria um único bean para toda a aplicação, e todas as solicitações para esse bean retornam a mesma instância.
- **Prototype:** Um bean com escopo prototype retorna uma nova instância sempre que for solicitado. Ele é útil quando não queremos compartilhar o estado do bean.
- **Request:** Disponível apenas em contextos web, cria um bean para cada requisição HTTP. Cada requisição recebe uma instância separada.
- **Session:** Disponível em contextos web, cria um bean por sessão HTTP. Durante uma mesma sessão, a mesma instância do bean é retornada.
- **Application:** Um bean com escopo application é único para o contexto da aplicação web (servidor de servlet). Ele é compartilhado entre todas as requisições e sessões.

Injetando Dependências

Agora, vamos falar um pouquinho sobre a injeção de dependências diretamente no Spring. É possível fazê-la de diversas formas, como através de arquivos XML, anotações padrões do Java ou por meio de classes de configuração. Mas vamos explorar primeiramente um dos pontos principais do *framework*, a anotação `@Autowired`.

A anotação **`@Autowired`** é um dos principais recursos do Spring para promover a injeção de dependências de forma simples, declarativa e minimamente invasiva. Ela **informa ao container do Spring que determinados pontos do código** – sejam eles construtores, métodos setter ou campos de classe – **devem receber objetos injetados automaticamente**, sem que o desenvolvedor precise se preocupar em instanciar manualmente essas dependências.

Seu uso se dá de três formas:

- **Construtores:** é considerado a abordagem mais recomendada porque torna as dependências obrigatórias explícitas no momento da criação do objeto. O Spring injeta os *beans* necessários diretamente no construtor da classe.

```java
@Component
public class PedidoService {
    private final PagamentoService pagamentoService;

    @Autowired
    public PedidoService(PagamentoService pagamentoService) {
        this.pagamentoService = pagamentoService;
    }
}
```

- **Métodos *setter*:** permite configurar dependências após a criação do objeto. Geralmente usada para dependências opcionais, onde o objeto pode funcionar mesmo sem a dependência injetada.

```java
@Component
public class PedidoService {
    private PagamentoService pagamentoService;

    @Autowired
    public void setPagamentoService(PagamentoService pagamentoService) {
        this.pagamentoService = pagamentoService;
    }
}
```

- **Campos de classe:** feita diretamente no atributo da classe, eliminando a necessidade de construtores ou métodos setter.

```java
@Component
public class PedidoService {
    @Autowired
    private PagamentoService pagamentoService;
}
```

Por padrão, o `@Autowired` tenta realizar a injeção pelo tipo do *bean*. Se existir exatamente um *bean* que corresponde ao tipo solicitado, a injeção é bem-sucedida. Caso haja mais de uma implementação possível, o Spring tentará resolver conflitos por meio de qualificadores ou do nome do *bean*. Nesse sentido, o uso combinado de anotações como `@Qualifier` ou a adoção de convenções de nomenclatura ajudam o container do Spring a resolver ambiguidade, selecionando corretamente a dependência a ser injetada.

Bom, a @Autowired, apesar de ser a principal, não é a única anotação para injeção de dependências – como havia lhe adiantado antes. Então fica aqui uma tabela-resumo com as principais anotações e outras formas de definirmos a injeção:

Método/Anotação	Descrição
▪ Arquivos XML	A definição dos *beans* e suas dependências é feita diretamente no arquivo de contexto.
@Autowired	Usada no Spring para injetar dependências automaticamente pelo tipo do *bean*. Funciona em construtores, setters e campos.
@Qualifier	Complementa o @Autowired, permitindo especificar qual *bean* deve ser injetado quando há múltiplos candidatos.
@Inject	Define a injeção de dependências seguindo a especificação JSR-330. Similar ao @Autowired.
@Named	Usada com @Inject para identificar *beans* pelo nome, semelhante ao uso de @Qualifier.
@Resource	Realiza a injeção de dependências baseada no nome do *bean*, conforme especificação JSR-250.
@Value	Injeta valores simples (string, números, booleanos) a partir de propriedades externas ou constantes.
@Bean	Define um método em uma classe de configuração (@Configuration) que retorna um *bean* gerenciado pelo container.
@Primary	Marca um *bean* como a implementação padrão para injeção quando há múltiplos candidatos disponíveis.

@Lookup	Injeta *beans* com escopo protótipo em componentes singleton, resolvendo a dependência em tempo de execução.
@DependsOn	Define a ordem de inicialização entre beans, garantindo que um bean seja inicializado antes de outro.

(UFRJ/TAE UFRJ/2023) A injeção de dependência é uma técnica de design usada para obter a inversão de controle. O Spring Framework oferece um recurso de injeção de dependência que permite aos objetos definir suas próprias dependências que o contêiner Spring posteriormente injeta nelas. Assinale a alternativa que NÃO faz parte dos recursos de injeção de dependência do Spring mais recente.

a) @Autowired
b) @Ioc
c) @Qualifier
d) @DependsOn
e) @Inject

Comentários:

Da lista apresentada, apenas @Ioc não é uma anotação válida. *(Gabarito: Letra B)*

Contexto

O **contexto** no Spring é o núcleo que **gerencia o ciclo de vida dos objetos** da aplicação, os *beans*. Ele é a implementação mais avançada do conceito de Inversão de Controle (IoC) e um dos elementos centrais do framework. O contexto é responsável por **instanciar, configurar, gerenciar e fornecer beans aos componentes que dependem deles**, resolvendo automaticamente as dependências e permitindo que o desenvolvedor foque na lógica de negócio, enquanto o Spring lida com os detalhes da infraestrutura.

O contexto no Spring é o núcleo que gerencia o ciclo de vida dos objetos da aplicação, conhecidos como beans. Ele é a implementação mais avançada do conceito de Inversão de Controle (IoC) e um dos elementos centrais do framework. O contexto é responsável por instanciar, configurar, gerenciar e fornecer beans aos componentes que dependem deles, resolvendo automaticamente as dependências e permitindo que o desenvolvedor foque na lógica de negócio, enquanto o Spring lida com os detalhes da infraestrutura.

Existem várias implementações do contexto no Spring, cada uma com objetivos específicos:

- **BeanFactory:** É a implementação mais básica do container IoC. Ele carrega e gerencia beans de forma preguiçosa (lazy loading), criando instâncias somente quando necessário.
- **ApplicationContext:** É uma extensão mais avançada do BeanFactory, oferecendo recursos adicionais como suporte a eventos, internacionalização, injeção de dependências automáticas e carregamento de beans por anotações. Implementações comuns incluem:
  - **ClassPathXmlApplicationContext:** Carrega configurações a partir de arquivos XML no classpath.
  - **FileSystemXmlApplicationContext:** Carrega configurações de arquivos XML a partir de um caminho no sistema de arquivos.
  - **AnnotationConfigApplicationContext:** Configura o contexto a partir de classes anotadas com `@Configuration`.
  - **WebApplicationContext:** Uma extensão especializada para aplicações web.

A configuração do contexto pode ser feita através da configuração do arquivo XML, a partir de anotações e baseado em classes, com a anotação `@Configuration`. Quanto às anotações, temos as seguintes:

Método/Anotação	Descrição
@Configuration	Indica que a classe define beans a serem gerenciados pelo Spring. Usada para configurações baseadas em Java.
@ComponentScan	Configura o Spring para escanear pacotes específicos em busca de componentes anotados como beans (@Component, @Service, etc.).
@Bean	Define explicitamente um bean gerenciado pelo container dentro de uma classe anotada com @Configuration.
@PropertySource	Especifica arquivos de propriedades externos a serem carregados no contexto, como application.properties.
@Import	Permite importar outras classes de configuração (@Configuration) no contexto atual.
@ImportResource	Carrega arquivos de configuração XML e os incorpora ao contexto Spring.
@Profile	Configura beans ou classes para serem carregados apenas em perfis específicos (como "dev", "prod").
@Scope	Define o escopo de um bean, como singleton, protótipo, request, session, entre outros.
@Lazy	Marca um bean para ser carregado de forma preguiçosa (lazy loading), somente quando for solicitado.
@Primary	Indica qual bean deve ser considerado prioritário quando há múltiplos candidatos para injeção.

Spring Expression Language (SpEL)

A **SpEL (Spring Expression Language)** é uma **linguagem de expressões** embutida no ecossistema do Spring, projetada para **avaliar e manipular propriedades, objetos e valores em tempo de execução**.

Enquanto linguagens de expressão existem para simplificar a configuração de aplicações e eliminar código repetitivo ou hard-coded, a SpEL se destaca por ser profundamente integrada aos mecanismos centrais do Spring, permitindo que desenvolvedores utilizem expressões tanto em configurações XML, anotações Java e propriedades externas, quanto em componentes internos da aplicação.

A SpEL segue uma sintaxe expressiva e clara, inspirada parcialmente em outras linguagens de expressão, como a Unified EL do Java EE. Ela oferece recursos como acesso a propriedades de objetos usando a notação ponto, invocação de métodos, chamadas a funções definidas no contexto, manipulação de coleções, avaliações condicionais e integrações nativas com o contexto Spring, incluindo acesso a *beans* gerenciados pelo container.

Além disso, a SpEL permite **resolver referências dinâmicas em tempo de execução**, interagindo com variáveis do ambiente, permitindo combinar valores configurados externamente e ajustá-los conforme o perfil ou ambiente de execução da aplicação, sem exigir mudanças no código-fonte principal.

Em termos de implementação, a SpEL é executada por um parser interno do Spring, e a avaliação das expressões é feita por meio de um `ExpressionParser` e de um `EvaluationContext`, que fornece o contexto de avaliação, incluindo variáveis, funções e referências a *beans*. Veja um exemplo de uso do SpEL:

Java
```java
analisador ExpressionParser = new SpelExpressionParser();
Expressão exp = parser.parseExpression("'Hello World'");
String mensagem = (String) exp.getValue();
```

(VUNESP/ALESP/2022) Na Linguagem de Expressão do Spring Framework (Spring Expression Language, SpEL), uma String literal é delimitada por

a) aspas duplas.
b) sinais de menor e maior.
c) parênteses.
d) chaves.
e) aspas simples.

Comentários:

Na SpEL, escrevemos uma string literal a partir de pares de aspas simples, como no exemplo anterior – em “ ‘Hello World’ “. *(Gabarito: Letra E)*

Spring Data Access e Integration

O **Spring Data Access/Integration** é a camada do ecossistema Spring projetada para **simplificar e padronizar o acesso a dados e a comunicação entre sistemas**. Ele oferece um conjunto de abstrações e ferramentas que tornam o desenvolvimento de aplicações corporativas mais eficiente, eliminando a necessidade de escrever código repetitivo e complexo ao lidar com bancos de dados, transações, frameworks de ORM, sistemas de mensageria e outros mecanismos de integração. Essa abordagem de alto nível não apenas reduz o acoplamento entre componentes, mas também facilita o teste, a manutenção e a escalabilidade das aplicações.

Uma das grandes forças do Spring Data Access/Integration é a consistência na maneira de lidar com recursos de persistência. Tradicionalmente, ao trabalhar com JDBC puro, por exemplo, os desenvolvedores precisavam gerenciar explicitamente conexões, criar statements, tratar exceções SQL específicas e fechar recursos manualmente. Com o Spring, todo esse *boilerplate* é abstraído. O framework fornece classes utilitárias, templates e mecanismos de exceções unificadas (como a `DataAccessException`) que tornam o código mais limpo, conciso e resiliente.

Além disso, o Spring Data Access é fortemente integrado com frameworks de mapeamento objeto-relacional (ORM), tais como o Hibernate, o JPA, o iBatis (MyBatis) e o JDO. Em vez de cada desenvolvedor aprender em detalhes a API do framework ORM escolhido, **o Spring oferece uma camada de abstração única**, padronizando o tratamento de transações, exceções e padronizando a interação com o contexto de persistência. Isso significa que podemos facilmente trocar de provedor ORM ou modificar a estratégia de persistência com mudanças mínimas no código.

Spring Data

O **Spring Data** é um projeto guarda-chuva dentro do ecossistema Spring que **visa simplificar o acesso a dados em diferentes tipos de armazenamentos**, sejam eles bancos de dados relacionais, bancos NoSQL ou mesmo outras fontes. Ele fornece uma camada de abstração que reduz drasticamente a quantidade de código repetitivo, eliminando a necessidade de escrever implementações complexas para operações de CRUD, paginação, ordenação e consultas customizadas. O objetivo principal do Spring Data é permitir que concentremos nossos esforços na lógica de negócio, sem ter que lidar diretamente com a infraestrutura de acesso a dados.

Uma das integrações mais populares é com o JPA (Java Persistence API) através do Spring Data JPA, permitindo que criemos repositórios baseados em interfaces, os chamados **Repositórios do Spring Data**, que se encarregam das operações de persistência. Ao definir uma interface que estende, por exemplo, JpaRepository, automaticamente herdamos métodos para salvar, atualizar, deletar e pesquisar entidades. Podemos ainda definir métodos de consulta usando convenções de nomes, e o Spring Data JPA analisará a assinatura desses métodos para gerar as queries subjacentes. Por exemplo, um método findByNome(String nome) em um repositório de Usuário pode gerar automaticamente uma consulta SELECT u FROM Usuario u WHERE u.nome = :nome.

As principais anotações aqui são:

Método/Anotação	Descrição
@Entity	Marca uma classe como uma entidade JPA, indicando que seus objetos serão persistidos no banco de dados.
@Id	Indica o atributo de uma entidade que representa a chave primária da tabela no banco de dados.
@GeneratedValue	Define a estratégia para geração do valor da chave primária (ex: autoincremento, sequência).
@Repository	Marca um componente da camada de acesso a dados, permitindo a tradução de exceções de banco para exceções Spring.
@EnableJpaRepositories	Habilita a detecção automática de repositórios JPA, indicando ao Spring onde escanear essas interfaces.
@Query	Permite definir consultas personalizadas (JPQL ou SQL nativo) diretamente na assinatura do método do repositório.
@Param	Associa parâmetros de método aos parâmetros definidos na consulta de um método anotado com @Query.

O ponto de partida no Spring Data é criar uma interface que estenda uma das interfaces de repositório padrão fornecidas pelo Spring Data, como o `CrudRepository`, `PagingAndSortingRepository` ou `JpaRepository` (quando utilizamos JPA). O JpaRepository, por exemplo, herda todos os métodos de `CrudRepository` e `PagingAndSortingRepository`, adicionando funcionalidades extras. Os métodos herdados permitem operações básicas: buscar por ID, listar todos, salvar, deletar, verificar existência, etc. Esses métodos já estão implementados pelo Spring Data, de forma que não precisamos escrever código adicional.

A interface do repositório é tipicamente parametrizada com dois tipos: a classe da entidade que ela gerenciará e o tipo da chave primária dessa entidade. Por exemplo, suponhamos que tenhamos uma entidade Usuario cuja chave primária seja do tipo Long. Criamos, então, uma interface para o repositório dessa entidade:

Java

```java
public interface UsuarioRepository extends JpaRepository<Usuario, Long> {
}
```

Com esse simples trecho de código, o Spring Data já oferece uma série de funcionalidades prontas. Ao iniciar a aplicação, o framework cria uma implementação concreta dessa interface em tempo de execução e a registra no contexto, possibilitando que, em qualquer ponto do código, possamos injetar (`@Autowired`) uma instância do `UsuarioRepository` e utilizá-la para persistir ou recuperar objetos do tipo Usuario.

Spring Integration

O uso do Spring incentiva o uso de interfaces e da injeção de dependência, para lidar com os Plain Old Java Object (POJO) e suas dependências necessárias. O Spring Integration leva esse conceito um pouco além, fazendo com que os POJOs estejam conectados em conjunto, usando um **paradigma de mensagem** para intercomunicação entre componentes.

Ele é voltado para facilitar a integração entre sistemas e componentes de software, oferecendo uma abordagem coerente e declarativa para lidar com fluxos de mensagens, transformação de dados e coordenação de processos. A ideia central do Spring Integration é **tratar a comunicação entre sistemas como um fluxo de mensagens trafegando por um conjunto de canais e endpoints**. Cada endpoint executa uma função específica, como rotear, transformar, filtrar, agrupar ou dividir mensagens. O uso de padrões bem definidos torna os processos de integração mais claros, testáveis e fáceis de manter, além de permitir que o desenvolvedor se concentre na lógica de negócio em vez de lidar diretamente com protocolos e APIs complexas.

As **mensagens são objetos que carregam um payload** (o dado propriamente dito) e metadados adicionais (headers) que ajudam no roteamento e tratamento. Essas mensagens trafegam por canais, que são abstrações de enfileiramento ou conexão entre endpoints. A partir desse modelo, é possível construir integrações complexas com uma gama de adaptadores e conectores pré-fornecidos para sistemas de arquivos, e-mails, protocolos de mensageria (JMS, AMQP), serviços REST, WebSockets, bancos de dados, entre outros.

Por exemplo, podemos criar um fluxo em que as mensagens são lidas de uma pasta no sistema de arquivos, transformadas (por exemplo, convertendo um arquivo texto em objeto JSON), roteadas para diferentes endpoints conforme seu conteúdo, enfileiradas em um broker de mensagens e, por fim, enviadas para um serviço REST externo.

A configuração dessa integração pode ser feita a partir de XML, anotações ou por uma abordagem baseada em Java DSL (Domain Specific Language), que permite escrever configurações de forma fluida e com proteção a tipagem, criando canais, endpoints e rotas por métodos encadeados, veja um exemplo:

```java
@EnableIntegration
@Configuration
public class IntegracaoConfig {
    @Bean
    public IntegrationFlow exemploFluxo() {
        return IntegrationFlows.from(Files.inboundAdapter(new
File("/entrada"))
                    .patternFilter("*.txt"))
                .transform(Files.toStringTransformer())
                .handle("processadorService", "processar")
                .channel("saida")
                .get();
    }
}
```

Nesse exemplo, um fluxo (`IntegrationFlow`) é definido. Ele lê arquivos .txt da pasta /entrada, transforma seu conteúdo em string, delega o processamento a um serviço qualquer (`processadorService`) e envia o resultado para um canal de saída. Cada etapa é um componente do Spring Integration, permitindo uma clara separação de responsabilidades e favorecendo a manutenção do código.

As anotações importantes aqui são:

Método/Anotação	Descrição
@EnableIntegration	Habilita os recursos de integração do Spring, permitindo a configuração de fluxos, canais e endpoints por meio de Java DSL ou outras abordagens.
@Transformer	Anota um método que recebe uma mensagem como entrada e a transforma (modifica ou converte seu conteúdo) antes de enviá-la para o próximo endpoint.
@ServiceActivator	Marca um método que atuará como um endpoint de serviço, processando mensagens recebidas de um canal e produzindo uma saída. É útil para lógica de negócios diretamente no fluxo.
@Filter	Indica que o método deve atuar como um filtro, decidindo se uma mensagem deve seguir adiante no fluxo ou ser descartada, com base em critérios definidos no próprio método.
@Router	Define um método que roteia mensagens para diferentes canais de saída, escolhendo o canal apropriado com base no conteúdo ou nos headers da mensagem.
@Splitter	Identifica um método que recebe uma única mensagem e a divide em várias mensagens menores, cada uma enviada separadamente para o próximo estágio do fluxo.

@Aggregator	Indica um método que combina várias mensagens relacionadas em uma única mensagem de saída, consolidando dados fragmentados ou resultados parciais.
@InboundChannelAdapter	Utilizada para criar um adaptador de canal de entrada, que gera mensagens a partir de uma fonte externa (como um diretório de arquivos ou um cron) e as envia para o canal configurado.

Spring Web

O **Spring Web** é um módulo do ecossistema Spring **focado em simplificar e estruturar o desenvolvimento de aplicações web e serviços REST**. Ele fornece um conjunto de componentes e ferramentas para lidar com requisições HTTP, mapear rotas para controladores, processar dados, aplicar validações, tratar erros e enviar respostas de forma coerente e padronizada. Com o Spring Web, a criação de aplicações web dinâmicas em Java se torna mais produtiva, limpa e organizada, permitindo que o desenvolvedor se concentre na lógica de negócio em vez de perder tempo reescrevendo código repetitivo.

Um dos conceitos centrais do Spring Web é o **DispatcherServlet**, um servlet especializado fornecido pelo framework que atua como um **"controlador frontal" (front controller)**. Ele recebe todas as requisições HTTP da aplicação, encaminha essas requisições para o controlador apropriado e, depois, retorna as respostas adequadas ao cliente. Esse padrão de design torna o fluxo da aplicação mais centralizado e previsível, facilitando a manutenção e o entendimento do código.

Para criar controladores, o Spring Web utiliza anotações como `@Controller` e `@RestController`. Os controladores são classes responsáveis por receber as requisições, processar a lógica de negócio (geralmente delegando para serviços) e retornar dados ou páginas HTML. No caso de uma aplicação de páginas dinâmicas, podemos retornar nomes de views (como arquivos JSP ou templates Thymeleaf) que o Spring, ou outro mecanismo de apresentação, se encarregará de renderizar. Já ao trabalhar com APIs REST, é comum usar `@RestController`, que faz com que os métodos dos controladores retornem dados (geralmente em JSON) diretamente no corpo da resposta HTTP, sem a necessidade de páginas HTML.

O mapeamento de rotas também é simples. Com a anotação `@RequestMapping` e suas variações como `@GetMapping`, `@PostMapping`, `@PutMapping` e `@DeleteMapping`, podemos definir facilmente quais URLs serão atendidas por cada método do controlador, bem como os métodos HTTP (GET, POST, PUT, DELETE, etc.) suportados. Por exemplo:

Java
```java
@RestController
@RequestMapping("/usuarios")
public class UsuarioController {

    @GetMapping
    public List<Usuario> listarTodos() {
        // lógica de listagem de usuários
    }

    @PostMapping
    public Usuario criarUsuario(@RequestBody Usuario usuario) {
        // lógica de criação de novo usuário
    }
}
```

O Spring Web oferece também suporte a conversões e validações. Ao receber parâmetros de requisição, podemos usar anotações como `@RequestParam` para acessar parâmetros simples (como strings e

números), `@PathVariable` para extrair variáveis da própria URL, e `@RequestBody` para extrair o corpo da requisição, tipicamente formatado em JSON ou XML. Além disso, se utilizarmos a integração com o Bean Validation (JSR-380), podemos anotar os campos de um objeto com restrições como `@NotNull`, `@Size`, `@Email`, entre outras, garantindo que os dados recebidos sejam válidos e, caso não sejam, o Spring retorne automaticamente uma resposta de erro adequada. Vamos agrupar as anotações:

Método/Anotação	Descrição
@Controller	Marca uma classe como um controlador, permitindo o uso de métodos mapeados para rotas web. Normalmente retorna views HTML.
@RestController	Combina @Controller e @ResponseBody, indicando que todos os métodos da classe retornam dados diretamente no corpo da resposta, ideal para APIs REST.
@RequestMapping	Define a rota básica ou prefixo de URL para a classe ou métodos, permitindo configurar caminho, métodos HTTP e outros atributos.
@GetMapping	Mapeia requisições HTTP GET para um método específico do controlador.
@PostMapping	Mapeia requisições HTTP POST para um método específico do controlador.
@PutMapping	Mapeia requisições HTTP PUT para um método específico do controlador.
@DeleteMapping	Mapeia requisições HTTP DELETE para um método específico do controlador.
@PathVariable	Extrai um valor dinâmico do próprio caminho da URL e o vincula a um parâmetro do método do controlador.
@RequestParam	Extrai valores de parâmetros da query string da URL, vinculando-os a parâmetros do método do controlador.
@RequestBody	Mapeia o conteúdo do corpo da requisição para um objeto Java, geralmente usado para receber JSON ou XML.
@ControllerAdvice	Define uma classe para tratar exceções e comportamentos globais de controle, aplicada a todos os controladores da aplicação.
@ExceptionHandler	Em conjunto com @ControllerAdvice ou em um controlador específico, trata exceções mapeando-as para respostas HTTP adequadas.
@ResponseStatus	Anota métodos ou exceções para definir o código de status HTTP retornado, sem precisar definir explicitamente em cada resposta.
@CrossOrigin	Configura o suporte a requisições de origem cruzada (CORS), permitindo que clientes de outros domínios acessem a API.

(FGV/TJ AP/2024) Breno está criando um serviço REST, que será disponibilizado por meio de um aplicativo Spring Boot. Como ele já conhece o padrão do REST, criou um método de inclusão no controlador, tendo como parâmetro uma entidade do tipo gerenciado pelo serviço.

Para que o parâmetro receba corretamente os dados fornecidos pela requisição, no formato JSON, Breno irá utilizar nesse parâmetro a anotação:

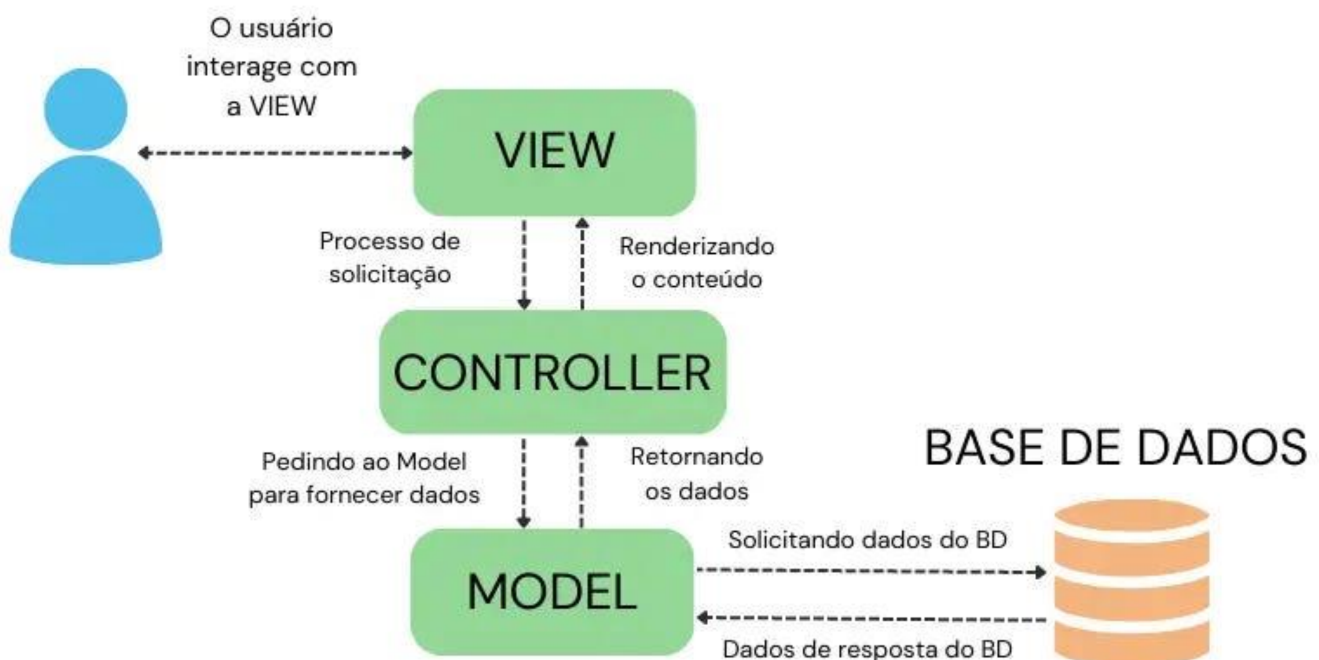
a) PostMapping;
b) RestController;
c) GetMapping;
d) RequestBody;
e) PathParam.

Comentários:

Os dados fornecidos fazem parte do corpo do pacote – sendo necessário usar, portanto, @RequestBody. *(Gabarito: Letra D)*

Spring MVC

O **Spring Web MVC** é um dos módulos mais importantes do ecossistema Spring, dedicado ao desenvolvimento de aplicações web seguindo o padrão Model-View-Controller (MVC). Esse padrão **separa claramente a aplicação em três camadas principais**: a camada de modelo (Model) responsável pelos dados e lógica de negócio, a camada de visão (View) encarregada da apresentação, e a camada de controle (Controller), que gerencia o fluxo de requisições e respostas. Essa arquitetura traz organização, facilitando a manutenção, os testes e a evolução da aplicação com o tempo.

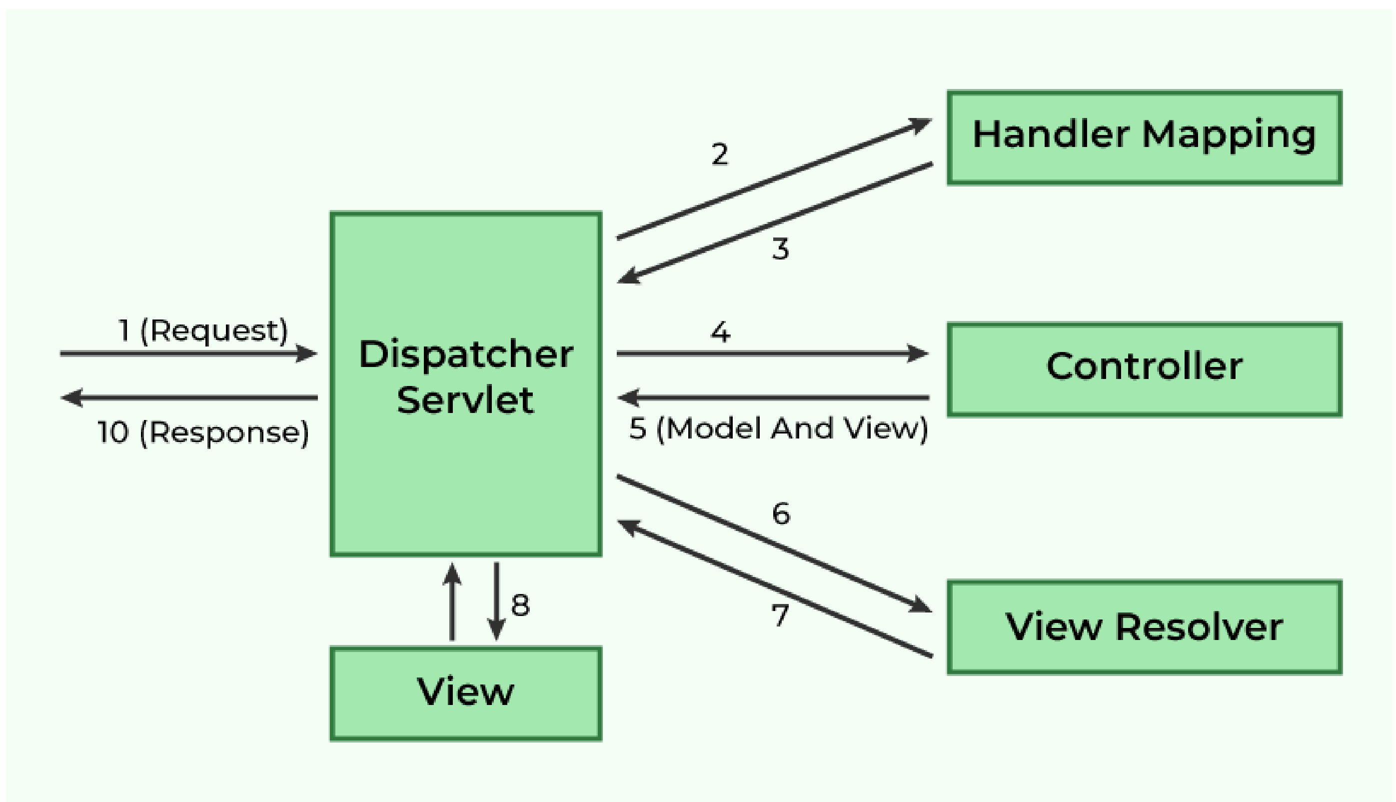


No centro do Spring Web MVC está o **DispatcherServlet**, um servlet especial configurado como front controller que recebe todas as requisições web. Ele delega a resolução da requisição a diversos componentes, como mapeadores de rotas (`HandlerMappings`) e adaptadores de manipulador (`HandlerAdapters`), para identificar qual controlador deve processar a solicitação.

Uma vez encontrado o controlador adequado, o DispatcherServlet chama seu método mapeado, que tipicamente interage com a camada de serviço ou o modelo para obter dados, processa-os, e retorna um resultado. Esse resultado pode ser um objeto simples, um nome lógico de view ou uma estrutura mais complexa (como um ModelAndView).

Para mapear métodos do controlador a rotas, o Spring MVC utiliza anotações como `@Controller` (ou `@RestController` para APIs REST) e `@RequestMapping` (ou suas variações como `@GetMapping`, `@PostMapping`, etc.). Essas anotações permitem declarar de forma simples qual URL e qual método HTTP correspondem a qual lógica de negócio. Ao ler essas anotações no início da aplicação, o Spring constrói internamente uma tabela de roteamento, tornando o fluxo de requisições altamente configurável e expressivo.

Após processar a requisição, o controlador retorna uma resposta que deve ser apresentada ao usuário. Se estivermos desenvolvendo um site dinâmico, é comum retornar um nome de view (como "listaUsuarios"), e então o ViewResolver buscará um template correspondente (por exemplo, um arquivo Thymeleaf ou JSP) para gerar a página HTML. Para aplicações REST, em que a resposta é geralmente JSON ou XML, o Spring MVC integra-se com HttpMessageConverters para transformar automaticamente objetos Java em representações textuais adequadas, enviando-as diretamente no corpo da resposta HTTP.



Em geral, temos os seguintes *beans*:

- **HandlerMapping:** responsável por mapear as requisições para determinados controladores ou métodos de controlador. Ele examina a URL, o método HTTP ou outras informações associadas à requisição e identifica qual "handler" a atenderá. Para o DispatcherServlet, o HandlerMapping

responde essencialmente à pergunta: "Dada esta requisição, qual controlador e método devem ser invocados?"

- **HandlerAdapter:** atua como um adaptador que sabe invocar o tipo específico de controlador retornado pelo HandlerMapping. Dado que o Spring suporta diferentes tipos de controladores (baseados em anotações, interfaces mais antigas, controladores assíncronos), o HandlerAdapter garante a integração entre o DispatcherServlet e o método do controlador. Ele lida com a conversão de argumentos, injeção de parâmetros da requisição e tratamento do objeto de resposta.
- **ViewResolver:** depois que o controlador processa a requisição e retorna um resultado (que pode ser um objeto, um nome lógico de view ou mesmo um objeto ModelAndView), o DispatcherServlet precisa determinar como gerar a resposta para o cliente. É aí que entra o ViewResolver. Esse componente mapeia o nome lógico da view para um objeto concreto de view, por exemplo, um template ou outro tipo de resposta (como JSON).
- **HandlerExceptionResolver:** em caso de erros ou exceções lançadas durante o processamento de uma requisição, o DispatcherServlet delega a resolução de erros aos HandlerExceptionResolvers. Eles analisam a exceção lançada e decidem qual resposta HTTP retornar ao cliente. Podem, por exemplo, mapear exceções específicas para códigos de status diferenciados (404, 400, 500), retornar páginas de erro personalizadas ou mesmo objetos JSON descrevendo o erro.
- **HandlerInterceptor:** Embora não seja um resolver ou mapper propriamente dito, o HandlerInterceptor é outro componente frequentemente usado ao lado do DispatcherServlet. Interceptores funcionam como filtros especializados que podem atuar antes ou depois de um método do controlador ser chamado, bem como após a view ser renderizada. Eles são úteis para implementar funcionalidades transversais, como autenticação, logging, medição de desempenho, monitoramento, entre outros.

(FGV/TJ MS/2024) No framework Spring MVC, o tipo de bean especial no WebApplicationContext que tem o objetivo de auxiliar o DispatcherServlet a invocar um manipulador mapeado para uma solicitação é o:

a) FlashMapManager;
b) HandlerAdapter;
c) HandlerMapping;
d) LocaleResolver;
e) MultipartResolver.

Comentários:

O *bean* especial que auxilia o Servlet a invocar um manipulador (controlador) específico é o HandlerAdapter. *(Gabarito: Letra B)*

Outro ponto de destaque do Spring Web MVC é a facilidade com que se lida com parâmetros, formulários e validações. Podemos extrair parâmetros diretamente da URL ou da query string usando anotações como `@RequestParam`, mapear partes dinâmicas do caminho da URL com `@PathVariable`, e obter o corpo de requisições JSON ou XML com `@RequestBody`. Caso precisemos validar esses dados, podemos anotar as classes do modelo com restrições (como @NotNull, @Size, @Email) e, usando o suporte a Bean Validation, o Spring MVC validará automaticamente as entradas, retornando respostas de erro coerentes se algo estiver inválido.

A separação entre o modelo, a lógica de controle e a apresentação faz com que testes se tornem mais simples. É possível testar controladores isoladamente, sem a necessidade de levantar um servidor ou conectar-se a um banco de dados real, graças ao contexto de testes do Spring. Além disso, interceptores (HandlerInterceptors) podem ser usados para adicionar comportamentos transversais, como auditoria, logging e autenticação, sem poluir o código de controle.

Para definirmos o Spring MVC nas dependências usando o XML, usamos:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

(Instituto Verbena/TJ AC/2024) O Spring MVC (Model-View-Controller) é um framework do Spring que facilita o desenvolvimento de aplicativos da web baseados no padrão de arquitetura MVC. Ele fornece um modelo de programação flexível e robusto para criar aplicativos da web, permitindo a separação clara de responsabilidades entre o modelo de dados (Model), a lógica de apresentação (View) e o controle de fluxo (Controller). Qual anotação no Spring MVC é usada para extrair valores de caminho (partes da URL) e passá-los para um método do controlador como argumentos?

a) @RequestHeader
b) @RequestParam
c) @ResponseBody
d) @PathVariable

Comentários:

Queremos mapear, conforme a questão, "valores de caminho (partes de URL)". Isso é feito a partir da anotação @PathVariable. *(Gabarito: Letra D)*

Spring WebFlux

O **Spring WebFlux** é um **framework reativo e não bloqueante** para o desenvolvimento de aplicações web, introduzido como parte do Spring 5, com o **objetivo de lidar de maneira mais eficiente com cenários de alta carga, grande número de requisições simultâneas e integrações que se beneficiam de fluxos de dados reativos**.

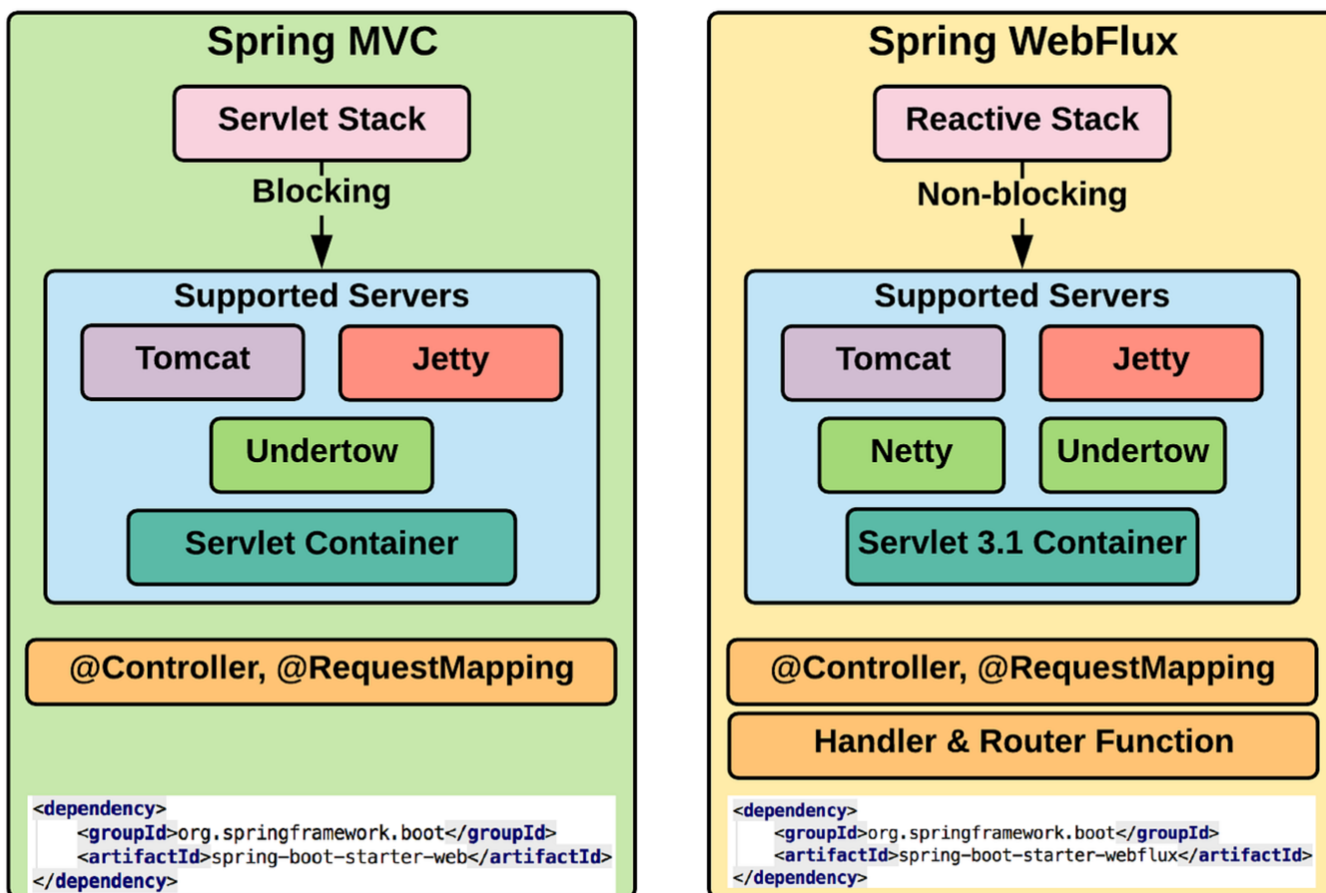
Em vez de seguir o modelo tradicional de threads bloqueantes do Spring Web MVC, o **WebFlux adota um paradigma assíncrono**, construído sobre reatores de eventos e conectores não bloqueantes, permitindo que a aplicação aproveite melhor os recursos do servidor e responda a mais solicitações com a mesma quantidade de hardware.

No modelo tradicional do Spring Web MVC, cada requisição geralmente associa uma thread do servidor (geralmente do Tomcat ou outro container de servlets), que fica bloqueada esperando a conclusão de operações como acesso a banco de dados ou serviços remotos. Isso pode ser eficiente para um número moderado de requisições, mas em cenários de altíssima escala, a contenção de threads e o bloqueio I/O tornam-se gargalos.

O **WebFlux**, por outro lado, **é construído sobre o Reactor**, uma biblioteca do ecossistema Spring que fornece uma API reativa baseada no padrão Reactive Streams e no modelo de programação reativa (inspirado no conceito de fluxos e observáveis). Em vez de bloquear a thread, as operações I/O de rede, acesso a dados ou chamadas a serviços externos são realizadas de forma assíncrona. Quando a resposta está pronta, o fluxo reativo avisa ao pipeline, que continua o processamento. Isso torna possível ao servidor lidar com milhares ou até milhões de conexões simultâneas com um número relativamente pequeno de threads, escalando horizontalmente de maneira mais eficiente.

A definição a partir do XML aqui funciona da seguinte forma:

```
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>
```



O Spring WebFlux oferece **dois estilos de programação** principais:

- **Estilo Anotado:** semelhante ao Spring MVC, podemos utilizar anotações como `@Controller` e `@GetMapping`, mas agora os métodos retornam tipos reativos como Mono<T> ou Flux<T> em vez de objetos simples. Um Mono<T> representa uma sequência reativa com zero ou um elemento, enquanto um Flux<T> representa uma sequência com zero, um ou vários elementos. O dispatcher do

WebFlux, diferentemente do DispatcherServlet tradicional, utiliza WebHandler e HandlerMapping adequados ao modelo reativo.

- **Estilo Funcional:** é uma abordagem baseada em roteamento e handlers funcionais. Em vez de anotações, definimos rotas e manipuladores usando uma DSL (Domain-Specific Language) em Java, criando funções que recebem uma requisição e retornam uma resposta. Esse estilo é mais próximo do modelo funcional e pode ser preferido por equipes que desejam evitar a complexidade de anotações, ou que querem um controle mais explícito do fluxo de dados. Veja um exemplo:

Java
```java
@Configuration
public class RouterConfig {

    @Bean
    public RouterFunction<ServerResponse> route(UserHandler handler) {
        return RouterFunctions.route(RequestPredicates.GET("/users"),
handler::getAllUsers)
                                   .andRoute(RequestPredicates.GET("/users/{id}")
, handler::getUserById);
    }
}
```

Aqui, definimos endpoints e ligamos cada rota a um método do handler, que, por sua vez, retorna um Mono<ServerResponse> ou Flux<ServerResponse>. Esse estilo elimina completamente o uso de `@Controller` ou `@RequestMapping`, adotando uma abordagem mais declarativa, próxima a frameworks funcionais.

Mesmo que o foco do WebFlux seja o reativo, o Spring Web MVC e o WebFlux não são mutuamente exclusivos. Na mesma aplicação (embora não seja um padrão comum em produção), é possível combinar ambos. Entretanto, na maior parte dos casos, escolhemos um stack: o stack síncrono (Web MVC, baseado em servlets) ou o stack reativo (WebFlux).

(CEBRASPE/PGDF/2021) Julgue o item seguinte, a respeito de JMS (Java Message Service), JUnit e Spring Framework.

O Spring WebFlux é compatível com Java 8 lambdas e Kotlin e tem a vantagem de permitir a criação de microsserviços com requisitos menos complexos.

Comentários:

De fato, o WebFlux possui compatibilidade com as funções lambda do Java 8 e Kotlin, sendo uma abordagem um pouco menos complexa para criar microsserviços. (Gabarito: Certo)

AOP

A **Programação Orientada a Aspectos (AOP)** é uma técnica que complementa a Programação Orientada a Objetos, permitindo **isolar e agrupar comportamentos que atravessam múltiplas camadas e classes de uma aplicação**, conhecidos como **concerns transversais**. Exemplos típicos de concerns transversais incluem logging, auditoria, segurança, transações e monitoramento de desempenho, entre outros. A ideia central é evitar que cada classe ou método precise explicitamente embutir lógicas repetitivas para lidar com essas questões, mantendo assim o código mais limpo e focado na regra de negócio principal.

No Spring, a AOP funciona por meio de um proxy gerado em tempo de execução, que intercepta as chamadas aos métodos anotados ou configurados. O container IoC do Spring, ao criar os beans, pode envolver determinados objetos com esses proxies, gerenciando a interceptação de chamadas e a injeção dos aspectos desejados, como interceptar todas as chamadas de métodos em determinada classe para fins de logging, por exemplo.

A AOP se baseia em alguns conceitos basilares:

- **Aspect:** É o módulo onde reunimos a lógica de um concern transversal. Em termos práticos, é uma classe anotada com `@Aspect`, contendo métodos que descrevem em quais pontos do código determinado comportamento deve ser "injetado".
- **Join Point:** É um ponto de execução dentro do fluxo da aplicação onde um aspecto pode ser aplicado. Por exemplo, a chamada de um método ou o lançamento de uma exceção. No Spring AOP, join points costumam ser a execução de métodos públicos em beans gerenciados pelo container.
- **Advice:** É o comportamento em si que será "injetado" no código, ou seja, a lógica que será executada no join point. O Spring fornece diferentes tipos de advice:
    - `@Before`: Executa antes do método interceptado.
    - `@After`: Executa após a conclusão do método (seja com sucesso ou exceção).
    - `@AfterReturning`: Executa somente após a conclusão bem-sucedida do método.
    - `@AfterThrowing`: Executa somente se o método lançar uma exceção.
    - `@Around`: Envolve a chamada do método interceptado, permitindo analisar e manipular a execução antes e depois do método em si.
- **Pointcut:** Especifica exatamente quais métodos ou classes devem ser interceptados por determinado advice. No Spring, costumamos usar expressões do tipo execution(...), within(...) e outros para filtrar métodos pelo pacote, pelas anotações, pelo modificador de acesso etc. A anotação `@Pointcut` pode encapsular essas expressões, tornando o código mais organizado.
- **Weaving (entrelaçamento):** É o processo de ligar o aspecto ao código de negócio no ponto determinado. No Spring, o weaving ocorre em tempo de execução por meio dos proxies dinâmicos gerados pelo container IoC. Não há, por padrão, modificação de bytecode em tempo de compilação.

Vamos explorar um exemplo.

```java
@Aspect
@Component
public class MonitoramentoAspect {

    @Pointcut("execution(* com.exemplo.servico.*.*(..))")
    public void metodosServico() {}

    @Around("metodosServico()")
    public Object medirTempoExecucao(ProceedingJoinPoint joinPoint) throws
Throwable {
        long inicio = System.currentTimeMillis();
        Object resultado = joinPoint.proceed();  // Continua a execução do
método interceptado
        long fim = System.currentTimeMillis();
        System.out.println("Tempo de execução do método "
                + joinPoint.getSignature() + ": " + (fim - inicio) + "ms");
        return resultado;
    }
}
```

No exemplo acima, temos alguns elementos importantes:

- `@Aspect`: Indica que essa classe é um aspecto.
- `@Pointcut("execution(* com.exemplo.servico.*.*(..))")`: Define que todos os métodos em qualquer classe do pacote com.exemplo.servico serão interceptados.
- `@Around("metodosServico()")`: Indica que o advice será executado "ao redor" do método, medindo o tempo antes e depois de sua execução.
- `ProceedingJoinPoint`: Representa o método interceptado, permitindo prosseguir com a execução usando proceed().

Ao anotar esse aspecto com `@Component`, o Spring irá encontrá-lo durante o scan de componentes. Para habilitar a AOP, normalmente utilizamos `@EnableAspectJAutoProxy` em uma classe de configuração ou na classe principal anotada com `@SpringBootApplication`. Esse recurso faz com que o container crie proxies quando detectar aspectos ativos, garantindo que o weaving ocorra em tempo de execução.

Testes

O **Spring Test** é o módulo de testes do Spring Framework. Ele oferece um conjunto de utilitários, anotações e funcionalidades para simplificar a criação e a execução de testes em aplicações baseadas no framework Spring. O objetivo é permitir que desenvolvedores validem o comportamento da lógica de negócio, da camada de persistência, da camada web e de configurações do contexto do Spring com o mínimo de esforço, mantendo a coerência com o ecossistema do framework.

O **Spring Test suporta tanto testes de unidade quanto testes de integração**. Em testes de unidade, o foco é verificar o comportamento de componentes específicos em isolamento, muitas vezes utilizando mocks para substituir dependências externas. Já os testes de integração, possibilitados pelo Spring Test, permitem carregar um contexto completo ou parcial do Spring, possibilitando a validação de interações reais entre componentes, acesso a banco de dados, transações, rotas web, segurança, entre outros aspectos.

Para habilitar os recursos do Spring Test, costuma-se utilizar o JUnit - até a versão 4, anotado com `@RunWith(SpringRunner.class)` e, em versões mais recentes, com o JUnit 5, integrando-se via `@ExtendWith(SpringExtension.class)`. A utilização dessas extensões do JUnit permite que o Spring inicialize o contexto da aplicação antes da execução dos métodos de teste, injetando dependências nos objetos de teste e permitindo o uso de anotações e funcionalidades específicas.

O Spring Test suporta a anotação `@ContextConfiguration` (ou `@SpringBootTest` no caso do Spring Boot) para especificar quais classes de configuração, perfis de ambiente e arquivos de propriedades devem ser carregados. Esse recurso garante que o contexto do Spring seja disponibilizado ao teste, permitindo a injeção de beans, verificação de transações e acesso a recursos que seriam utilizados em produção.

Com o contexto disponível, podemos usar `@Autowired` para obter instâncias reais de beans, reduzindo a necessidade de criar mocks ou stubs manualmente. Isso é particularmente útil em testes de integração, onde se deseja verificar a integração entre a camada de serviço e a camada de persistência, por exemplo.

Anotações

As anotações são a parte mais cobrada do Spring. Já vimos várias, vamos reunir elas e outras aqui numa tabela-resumo para que você chegue preparado na hora da prova.

Método/Anotação	Descrição
@Component	Marca uma classe como um bean gerenciado pelo Spring (Core Container). É detectada via varredura de componentes (@ComponentScan).
@Service	Especialização de @Component. Indica que a classe representa a camada de serviço na aplicação, concentrando lógica de negócios.
@Repository	Especialização de @Component. Usada para a camada de acesso a dados. O Spring pode traduzir exceções específicas de banco para exceções genéricas de acesso a dados.
@Controller	Indica que a classe atua como controlador na camada web (Spring MVC). Geralmente retorna nomes lógicos de views (HTML, JSP, etc.) ao invés de dados brutos.
@RestController	Combina @Controller + @ResponseBody. Usada em APIs RESTful. Os métodos retornam dados (JSON, XML) diretamente no corpo da resposta.
@Configuration	Indica que a classe define beans e configurações do Spring. Geralmente usada junto com métodos @Bean.
@Bean	Em uma classe @Configuration, define um método que instancia, configura e retorna um objeto, que passa a ser gerenciado como bean no container IoC.
@Scope	Especifica o escopo do bean (singleton, prototype, request, session etc.).
@Qualifier	Diferencia beans do mesmo tipo na injeção de dependência, definindo qual bean deve ser escolhido quando há mais de um candidato.
@Primary	Quando vários beans do mesmo tipo estão disponíveis, indica qual deve ser considerado prioritário durante a injeção (substitui a necessidade de @Qualifier).
@Value	Injeta valores simples (strings, números, booleans) a partir de configurações externas, como arquivos .properties ou variáveis de ambiente.
@Autowired	Injeção de dependência automática baseada em tipo. Pode ser aplicada em construtores, setters ou campos de classe.
@Lazy	Indica que a criação do bean deve ser adiada (lazy loading) até o momento de sua primeira utilização.

@Required (obsoleto)	Indicava que uma propriedade obrigatória deveria ser injetada; caso contrário, lançaria exceção. Nas versões atuais, o uso foi desaconselhado em favor de construtores obrigatórios ou validações.
@DependsOn	Garante que um bean seja inicializado somente após a inicialização de outro bean especificado.
@Profile	Permite definir em quais perfis (por ex.: dev, test, prod) determinado bean ou configuração deve ser carregado.
@PostConstruct (JSR-250)	Executa um método logo após a injeção de dependências e inicialização do bean.
@PreDestroy (JSR-250)	Executa um método antes da destruição do bean, permitindo liberações de recursos.
@Conditional	(Introduzido no Spring 4) Permite criar beans ou comportamentos condicionais com base em uma expressão programática ou em implementações de Condition. (Subanotações específicas ao Boot foram removidas.)
@ComponentScan	Especifica em quais pacotes o Spring deve procurar classes anotadas como componentes (@Component, @Service, @Repository, etc.).
@Aspect	Marca uma classe como um aspecto para AOP. Contém métodos advice e pointcuts que serão aplicados a pontos específicos do código.
@EnableAspectJAutoProxy	Habilita a detecção e criação automática de proxies para as classes anotadas com @Aspect, possibilitando weaving em tempo de execução.
@Before	Advice que executa antes do método interceptado (AOP).
@After	Advice que executa após o método interceptado, independentemente do resultado (exceção ou não).
@AfterReturning	Advice que executa somente depois da conclusão bem-sucedida do método interceptado.
@AfterThrowing	Advice que executa caso o método interceptado lance uma exceção.
@Around	Advice que envolve completamente a chamada do método (antes e depois). Permite controle avançado do fluxo, inclusive a possibilidade de alterar o retorno.
@Transactional	Declara que um método ou classe deve ser executado dentro de um contexto transacional, simplificando o gerenciamento de transações com a camada de persistência.
@EnableTransactionManagement	Habilita o suporte a transações declarativas por meio de @Transactional.

@Entity (JPA)	Marca uma classe como entidade persistente, mapeando-a para uma tabela no banco de dados.
@Id (JPA)	Indica o atributo da entidade que funciona como chave primária.
@GeneratedValue (JPA)	Especifica a estratégia de geração de valor para a chave primária (ex.: AUTO, IDENTITY, SEQUENCE).
@Table (JPA)	Configura o nome da tabela e outras características, caso sejam diferentes dos padrões inferidos.
@Column (JPA)	Especifica detalhes de mapeamento de colunas, como nome, nullability e tamanho.
@Repository (Spring Data)	Além do uso geral, no contexto do Spring Data indica uma interface de repositório (ex.: UserRepository) que estende interfaces como JpaRepository, possibilitando consultas derivadas ou customizadas.
@EnableJpaRepositories	Habilita e configura a varredura de repositórios JPA, detectando interfaces que estendem JpaRepository ou outras.
@Query (Spring Data JPA)	Define consultas personalizadas (JPQL ou SQL nativo) diretamente na interface de repositório.
@Param (Spring Data JPA)	Associa parâmetros de métodos em repositórios aos parâmetros da query definida em @Query.
@NoRepositoryBean (Spring Data)	Indica que uma interface não deve ser tratada como repositório concreto, servindo apenas como base para outras interfaces que estenderão seus métodos.
@EnableWebMvc	Ativa a configuração padrão do Spring MVC e permite customizações avançadas via WebMvcConfigurer.
@RequestMapping	Mapeia solicitações HTTP para classes ou métodos de controlador (também há variações como @GetMapping, @PostMapping, @PutMapping, @DeleteMapping).
@GetMapping / @PostMapping etc.	Especificam o método HTTP (GET, POST, PUT, DELETE) para rotas dentro de controladores MVC.
@RequestParam	Injeta o valor de parâmetros da query string em parâmetros do método do controlador.
@PathVariable	Extrai partes dinâmicas do caminho da URL, vinculando-as a parâmetros do método do controlador.
@RequestBody	Lê o corpo da requisição (geralmente JSON ou XML) e o converte em um objeto Java, injetado como argumento do método.
@ResponseBody	Retorna o objeto diretamente no corpo da resposta em vez de usar o mecanismo de view (combinado a @Controller).
@CrossOrigin	Configura o suporte a CORS (Cross-Origin Resource Sharing) para permitir ou restringir requisições de origens diferentes.

@ResponseStatus	Declara o código de status HTTP que deve ser retornado caso o método do controlador ou exceção anotada seja acionado.
@ControllerAdvice	Define uma classe global de configuração de tratamento de exceções ou atribuições para todos os controladores da aplicação (ex.: métodos com @ExceptionHandler).
@ExceptionHandler	Dentro de controladores ou de classes anotadas com @ControllerAdvice, permite mapear exceções a métodos que produzem respostas HTTP adequadas (ex.: 400, 404, 500).
@ModelAttribute	Vincula atributos do modelo à sessão ou a campos de formulário em controladores MVC.
@SessionAttributes	Define que determinados atributos do modelo devem ser mantidos na sessão HTTP, persistindo entre requisições do usuário.
@EnableWebFlux	Habilita a configuração base para aplicações reativas no Spring WebFlux, registrando handlers e adaptadores reativos.
@EnableScheduling	Habilita a execução de tarefas agendadas (scheduling) via anotação @Scheduled.
@Scheduled	Configura métodos para serem executados de forma periódica ou em intervalos específicos (cron jobs).
@EnableAsync	Habilita o processamento assíncrono em métodos anotados com @Async, permitindo execução em threads separadas.
@Async	Marca métodos para serem executados de forma assíncrona, liberando a thread que fez a chamada.
@EnableIntegration	Habilita recursos de integração (Spring Integration), permitindo a configuração de fluxos de mensagens, canais, adaptadores etc.
@Transformer / @Filter etc.	(Spring Integration) Anota métodos que transformam, filtram, roteiam ou processam mensagens em canais de integração, seguindo padrões de integração empresarial (EIP).
@EnableSecurity / @EnableWebSecurity	Habilitam a configuração de segurança, definindo regras de autenticação, autorização, filtros e interceptação de requisições.
@EnableCaching	Ativa o suporte a cache dentro do contexto Spring, possibilitando o uso de anotações como @Cacheable, @CacheEvict e @CachePut.
@Cacheable	Anota métodos cujos resultados devem ser armazenados em cache, evitando execuções repetidas e melhorando desempenho.
@CacheEvict	Remove entradas de cache após o método anotado ser executado, mantendo o cache coerente quando há alterações de dados.
@CachePut	Atualiza o cache com o resultado de um método, independente de já existir alguma entrada armazenada.

(FGV/TJ AP/2024) A analista Joelma está desenvolvendo o web service tjapRest utilizando o Spring Boot. Determinadas operações de tjapRest devem ser executadas assincronamente. Para implementar as operações assíncronas de forma simples e direta, Joelma recorreu a duas anotações padrões do Spring. A primeira anotação habilita o suporte do Spring à execução de métodos assíncronos. A segunda anotação marca determinado método como um candidato à execução assíncrona.

Joelma recorreu às anotações do Spring:

a) @EnableAsync e @Async;
b) @AsyncConfigurer e @Async;
c) @EnableAsync e @Asynchronous;
d) @AsyncConfigurer e @EnableAsync;
e) @AsyncConfigurer e @Asynchronous.

Comentários:

Para habilitarmos o assincronismo e marcamos os métodos à execução assíncrona, usamos, respectivamente, @EnableAsync e @Async. *(Gabarito: Letra A)*

QUESTÕES COMENTADAS

01. (INQC/CPTRANS/2024) Utilizando o framework Spring MVC, caso se deseje criar um controller, a anotação a ser utilizada no código é:

a) #Controller
b) &Controller
c) @Controller
d) %Controller

Comentários:

Vamos começar com uma questão tranquila – as anotações em Java são marcadas pelo arroba `@`. Para criar uma anotação do Controller, usamos `@Controller`.

Gabarito: Letra C

---

02. (FGV/TJ AP/2024) Breno está criando um serviço REST, que será disponibilizado por meio de um aplicativo Spring Boot. Como ele já conhece o padrão do REST, criou um método de inclusão no controlador, tendo como parâmetro uma entidade do tipo gerenciado pelo serviço.

Para que o parâmetro receba corretamente os dados fornecidos pela requisição, no formato JSON, Breno irá utilizar nesse parâmetro a anotação:

a) PostMapping;
b) RestController;
c) GetMapping;
d) RequestBody;
e) PathParam.

Comentários:

Queremos uma anotação para receber os dados da requisição em JSON. Vamos analisar as alternativas:

a) Errado. PostMapping: Mapeia requisições HTTP do tipo POST para um método específico do controlador, indicando que aquele método será chamado quando uma requisição POST for recebida na rota configurada.
b) Errado. RestController: Marca a classe como um controlador REST, fazendo com que os métodos retornem dados (normalmente em JSON ou XML) diretamente no corpo da resposta HTTP, em vez de retornarem páginas ou views.
c) Errado. GetMapping: Mapeia requisições HTTP do tipo GET para um método específico do controlador, definido para lidar com leituras ou consultas de recursos na rota indicada.
d) Certo. RequestBody: Associa o conteúdo do corpo da requisição (geralmente em JSON ou XML) a um parâmetro do método, convertendo os dados recebidos em um objeto Java que o método pode processar.

e) Errado. PathParam: No contexto de JAX-RS (e não do Spring MVC), especifica que o valor de um parâmetro de método deve ser obtido diretamente de uma parte do caminho (path) da URL, permitindo rotas com variáveis dinâmicas.

Portanto, correta a letra D.

Gabarito: Letra D

---

03. (FGV/TJ AP/2024) A analista Joelma está desenvolvendo o web service tjapRest utilizando o Spring Boot. Determinadas operações de tjapRest devem ser executadas assincronamente. Para implementar as operações assíncronas de forma simples e direta, Joelma recorreu a duas anotações padrões do Spring. A primeira anotação habilita o suporte do Spring à execução de métodos assíncronos. A segunda anotação marca determinado método como um candidato à execução assíncrona.

Joelma recorreu às anotações do Spring:

a) @EnableAsync e @Async;
b) @AsyncConfigurer e @Async;
c) @EnableAsync e @Asynchronous;
d) @AsyncConfigurer e @EnableAsync;
e) @AsyncConfigurer e @Asynchronous.

Comentários:

Queremos anotações para habilitar o suporte à execução assíncrona de métodos, e para marcar métodos para execução assíncrona. Usaremos, respectivamente, as anotações @EnableAsync e @Async.

Gabarito: Letra A

---

04. (FGV/TJ MS/2024) No framework Spring MVC, o tipo de bean especial no WebApplicationContext que tem o objetivo de auxiliar o DispatcherServlet a invocar um manipulador mapeado para uma solicitação é o:

a) FlashMapManager;
b) HandlerAdapter;
c) HandlerMapping;
d) LocaleResolver;
e) MultipartResolver.

Comentários:

Vamos analisar o que cada *bean* trazido pela questão:

- FlashMapManager: Gerencia dados efêmeros (Flash Attributes) que sobrevivem entre duas requisições HTTP consecutivas, normalmente utilizados em redirecionamentos para transferir mensagens ou informações temporárias.

- HandlerAdapter: Atua como um “adaptador” que sabe como invocar o método ou componente responsável pelo tratamento de uma requisição (handler), abstraindo o estilo de controlador utilizado (anotações, interfaces antigas etc.).
- HandlerMapping: Mapeia as requisições (URL, método HTTP, etc.) para o controlador ou método adequado, definindo qual “handler” lidará com determinada rota.
- LocaleResolver: Determina a localidade (idioma, formatação de data/número, etc.) para cada requisição, possibilitando a aplicação de internacionalização (i18n).
- MultipartResolver: Lida com requisições multipart/form-data, tipicamente usadas em uploads de arquivos, parseando o conteúdo para que o controlador possa acessar arquivos e campos enviados pelo cliente.

Portanto, o *bean* que irá auxiliar, invocando os métodos para o Servlet, é o HandlerAdapter.

Gabarito: Letra B

---

05. (Instituto Verbena/TJ AC/2024) O Spring MVC (Model-View-Controller) é um framework do Spring que facilita o desenvolvimento de aplicativos da web baseados no padrão de arquitetura MVC. Ele fornece um modelo de programação flexível e robusto para criar aplicativos da web, permitindo a separação clara de responsabilidades entre o modelo de dados (Model), a lógica de apresentação (View) e o controle de fluxo (Controller). Qual anotação no Spring MVC é usada para extrair valores de caminho (partes da URL) e passá-los para um método do controlador como argumentos?

a) @RequestHeader
b) @RequestParam
c) @ResponseBody
d) @PathVariable

Comentários:

Muitas vezes precisamos extrair partes da URL e passá-los como parâmetros para algum método do Controller. A anotação que é responsável por essa ação é a @PathVariable. Quanto às demais:

- @RequestHeader: Extrai valores do cabeçalho (header) HTTP da requisição e os vincula a parâmetros do método do controlador.
- @RequestParam: Mapeia parâmetros da query string (ou campos de formulário) para parâmetros do método, facilitando o acesso a valores enviados na URL.
- @ResponseBody: Indica que o valor de retorno do método deve ser escrito diretamente no corpo da resposta HTTP, em vez de usar a resolução de uma view (ex.: retorno JSON ou texto simples).

Portanto, correta a letra D.

Gabarito: Letra D

---

06. (COMPERVE/UFRN/2023) O Spring framework é uma ferramenta amplamente utilizada no desenvolvimento de aplicações Java Web. No ciclo de vida de um Spring Bean, é possível utilizar anotações em métodos que vão adicionar algum comportamento nos momentos de criação e na destruição desse Bean. As duas anotações utilizadas no spring para usar esses métodos customizados, no ciclo de vida dos Beans, são

a) @PostConstruct e @PreDestroy.
b) @PostConstruct e @BeforeDestroy.
c) @AfterConstruct e @PreDestroy.
d) @AfterConstruct e BeforeDestroy.

Comentários:

Para realizarmos comportamentos na criação e na destruição, usamos, respectivamente, @PostConstruct e @PreDestroy. O @PostConstruct é usada para marcar um método que deve ser executado após o bean ser inicializado pelo contêiner Spring e todas as dependências terem sido injetadas. Esse método é útil para executar lógicas de configuração ou inicialização específicas. Já o @PreDestroy é usado para marcar um método que será executado antes do bean ser destruído pelo contêiner Spring. Isso é útil para liberar recursos, fechar conexões ou realizar outras tarefas de limpeza.

Gabarito: Letra A

---

07. (COMPERVE/UFRN/2023) No Spring framework, o escopo de um Bean define a sua visibilidade e o seu ciclo de vida. Sobre os escopos do Spring Framework, analise as afirmativas abaixo, num contexto de uma aplicação web.

I Se nenhum escopo for especificado, o escopo padrão utilizado é o Application.
II O escopo Session retorna uma instância do Bean única sempre que for requerido.
III O escopo Prototype retorna uma instância diferente do Bean sempre que for requerido.
IV Se nenhum escopo for especificado, o escopo padrão utilizado é o Singleton.

Entre as afirmativas, estão corretas

a) III e IV.
b) I e II.
c) I e III.
d) II e IV.

Comentários:

No Spring Framework, o escopo de um bean define como e quando os objetos são criados e compartilhados. No contexto de uma aplicação web, os escopos disponíveis são:

- **Singleton (Padrão):** É o escopo padrão. Quando nenhum escopo é especificado, o Spring cria um único bean para toda a aplicação, e todas as solicitações para esse bean retornam a mesma instância.
- **Prototype:** Um bean com escopo prototype retorna uma nova instância sempre que for solicitado. Ele é útil quando não queremos compartilhar o estado do bean.

- **Request:** Disponível apenas em contextos web, cria um bean para cada requisição HTTP. Cada requisição recebe uma instância separada.
- **Session:** Disponível em contextos web, cria um bean por sessão HTTP. Durante uma mesma sessão, a mesma instância do bean é retornada.
- **Application:** Um bean com escopo application é único para o contexto da aplicação web (servidor de servlet). Ele é compartilhado entre todas as requisições e sessões.

Com isso em mente, vamos analisar as alternativas:

I. Errado. O escopo padrão do Spring é Singleton, não Application.
II. Errado. O escopo Session retorna uma única instância do bean por sessão HTTP, mas não para todas as requisições ou sessões.
III. Certo. O escopo Prototype cria uma nova instância do bean a cada solicitação.
IV. Certo. No Spring, o escopo padrão é o Singleton.

Portanto, corretos os itens III e IV.

Gabarito: Letra A

---

08. (CEBRASPE/TBG/2023)

```
package com.example.springboot;

import
org.springframework.web.bind.annotation.GetMappi
ng;
import
org.springframework.web.bind.annotation.RestCont
roller;

@RestController
public class HelloController {
    @GetMapping(“/”)
     public String index() {
         return “Greetings from Spring Boot!”;
     }
}
```

A partir do código precedente, julgue o item subsecutivo, relativo a Spring.

No Spring, a RestController é usada para marcar a classe como um controlador em que cada método retorna um objeto de domínio em vez de uma exibição.

Comentários:

No Spring, a anotação @RestController é utilizada para marcar uma classe como um controlador REST. A principal característica dessa anotação é que ela combina as funcionalidades de @Controller e @ResponseBody, o que significa que:

- Todos os métodos da classe anotada retornam diretamente dados no corpo da resposta HTTP (como JSON ou XML), em vez de retornar o nome de uma view para renderização de páginas HTML.
- O comportamento é ideal para a construção de APIs RESTful, onde os métodos retornam objetos de domínio, DTOs (Data Transfer Objects) ou respostas processadas, sem envolvimento de mecanismos de exibição.~

Portanto, o item está correto.

Gabarito: Certo

---

09. (PR4/UFRJ/2023) A injeção de dependência é uma técnica de design usada para obter a inversão de controle. O Spring Framework oferece um recurso de injeção de dependência que permite aos objetos definir suas próprias dependências que o contêiner Spring posteriormente injeta nelas. Assinale a alternativa que NÃO faz parte dos recursos de injeção de dependência do Spring mais recente.

a) @Autowired
b) @Ioc
c) @Qualifier
d) @DependsOn
e) @Inject

Comentários:

Vamos analisar as alternativas, procurando aquela que não faz parte do Spring.

a) Certo. Utilizada para injeção de dependência automática pelo tipo.
b) Errado. Não existe no Spring.
c) Certo. Complementa @Autowired para resolver ambiguidades quando há mais de um bean do mesmo tipo.
d) Certo. Indica que o bean anotado depende de outro bean ou beans específicos, garantindo que o(s) bean(s) dependente(s) seja(m) inicializado(s) primeiro.
e) Certo. Pertence à especificação JSR-330 e é equivalente funcionalmente ao @Autowired. É suportada pelo Spring Framework e também realiza a injeção de dependência baseada em tipo.

Portanto, incorreta a letra B.

Gabarito: Letra B

---

10. (FCC/TRT 21/2023) O RestTemplate é uma classe do Spring Framework usada para fazer chamadas HTTP a serviços externos. A anotação usada para modificar o comportamento padrão do RestTemplate indicando que ele deve ser configurado para suportar o balanceamento de carga ao chamar serviços registrados em um servidor de descoberta, como o Eureka, é a anotação

a) @LoadBalanced
b) @LoadBalancedRestfull
c) @ReatfullLoadBalanced

d) @Component Balanced
e) @BeanLoadBalanced

Comentários:

Para suportarmos balanceamento de carga, usamos a anotação @LoadBalanced. Ela é utilizada para configurar um RestTemplate ou um WebClient para suportar balanceamento de carga ao fazer chamadas a serviços registrados em um servidor de descoberta, como o Eureka. Quando aplicada, essa anotação indica ao Spring que ele deve usar um cliente HTTP com capacidades de balanceamento de carga, permitindo que o nome do serviço seja resolvido dinamicamente.

Gabarito: Letra A

---

11. (CEBRASPE/DP DF/2022) Julgue o próximo item, a respeito de Spring Framework, JDBC, iText, Java 8 e Apache CXF.

No Spring Framework, a anotação @RequestClass é usada para que a aplicação saiba a requisição que deve ser tratada pelo @Controller.

Comentários:

A anotação @RequestClass não existe no Spring Framework. No Spring Framework, a anotação utilizada para mapear requisições HTTP a métodos ou classes de um controlador é @RequestMapping (ou suas variações mais específicas, como @GetMapping, @PostMapping, @PutMapping e @DeleteMapping). Essas anotações permitem definir a URL e o método HTTP que um controlador ou método deve atender.

Gabarito: Errado

---

12. (VUNESP/ALESP/2022) Na Linguagem de Expressão do Spring Framework (Spring Expression Language, SpEL), uma String literal é delimitada por

a) aspas duplas.
b) sinais de menor e maior.
c) parênteses.
d) chaves.
e) aspas simples.

Comentários:

Para delimitarmos strings literais usamos aspas simples.

Gabarito: Letra E

---

13. (FGV/TJ TO/2022) O técnico em informática Marcos implementou o web service REST obtemPong utilizando Java com framework Spring. O web service obtemPong recebe o parâmetro obrigatório ping.

Observe abaixo o principal trecho do código-fonte de obtemPong:

```
@GetMapping("/api/v1/pong") @ResponseBody
public String obtemPong(@RequestParam String ping) {
                return ping;
}
```

Para que o parâmetro ping deixe de ser obrigatório e automaticamente assuma o valor “pong” caso esteja ausente da mensagem de requisição, Marcos deve adicionar à anotação @RequestParam do parâmetro ping o argumento:

a) value = “pong”;
b) name = “pong”;
c) defaultValue = “pong”;
d) required = “ping?ping:pong”;
e) value = “ping?ping:pong”.

Comentários:

No Spring Framework, a anotação @RequestParam possui o atributo defaultValue, que permite definir um valor padrão para o parâmetro caso ele não seja fornecido na requisição. Quando o valor padrão é especificado, o parâmetro deixa de ser obrigatório, e o Spring usa o valor especificado no defaultValue quando ele estiver ausente na requisição. Portanto, no contexto da questão, podemos definir defaultValue = “pong”;.

Gabarito: Letra C

---

14. (FCC/TRT 19/2022) Na abordagem do Spring para construir serviços web RESTful, as solicitações HTTP são tratadas por um controlador, que é uma classe identificada com a anotação

a) @RestfulController
b) @RestfulMapping
c) @RestApplicationController
d) @RestController
e) @RequestController

Comentários:

O controlador para serviços web REST é anotado com @RestController.

Gabarito: Letra D

---

15. (FCC/TRT 19/2022) Na abordagem do Spring para construir serviços web RESTful, as solicitações HTTP são tratadas por uma classe conhecida como controlador. Nessa classe, a anotação que garante que as requisições HTTP GET feitas para /dados sejam mapeadas para o método dados() é a anotação

a) @GetMappings("method=dados")
b) @RequestMappingMethod("/dados")

c) @GetMapping("/dados")
d) @GetMappingMethod("/dados")
e) @RequestMappings(method=dados)

Comentários:

Queremos mapear uma requisição GET para /dados – isso é feito através do grupo de anotações do @RequestMapping, mais especificamente o @GetMapping. Para especificarmos as requisições para /dados, usamos @GetMapping("/dados").

Gabarito: Letra C

---

16. (FCC/TRT 23/2022) A base do contêiner Inversion of Control (IoC), também conhecido como Dependency Injection (DI), do Spring Framework, é formada pelos pacotes

a) org.springframework.beans e org.springframework.context
b) org.springframework.orm e org.springframework.jdbc
c) org.springframework.web e org.springframework.webmvc
d) org.springframework.core e org.springframework.expression
e) org.springframework.webmvc e org.springframework.websocket

Comentários:

Para o contêiner de IoC, precisamos inserir dois pacotes – os *beans* e o contexto. Para isso, usamos o org.springframework.beans e o org.springramework.context.

Gabarito: Letra A

---

17. (IDECAN/TJ PI/2022) O Spring é um framework desenvolvido para a plataforma Java que facilita a vida do desenvolvedor quando falamos da construção de código de infraestrutura. Baseado na ideia da inversão de controle e injeção de dependência, Spring conta com diversos módulos que auxiliam na construção de aplicações corporativas.

A respeito dos conceitos e módulos presentes no framework, analise as afirmativas abaixo e marque alternativa correta.

I. No Spring a utilização da inversão de controle é facilitada graças à injeção de dependência.
II. @Autowired é a notação utilizada em Spring quando desejamos trabalhar com injeção de dependência por campo.
III. Spring Boot é um dos integrantes do framework do Spring. Tem foco na missão de facilitar o processo de configuração das aplicações. Essa facilitação ocorre graças ao conceito de convenção sobre a configuração.

a) Apenas as afirmativas I e II estão corretas.
b) Apenas as afirmativas I e III estão corretas.
c) Apenas a afirmativa I está correta.
d) Apenas as afirmativas II e III estão corretas.

e) Todas as afirmativas estão corretas.

Comentários:

Vamos analisar os itens.

> I. Certo. A IoC no Spring é feita a partir da injeção de dependências, realizada, dentre outros métodos, a partir da anotação @Autowired.
> II. Certo. Conforme vimos no item anterior.
> III. Certo. Esse item foge um pouco do que vimos, mas o Spring Boot faz parte do Spring Framework (e usualmente é visto em uma aula separada), buscando trazer agilidade e facilidade na configuração inicial.

Portanto, todos os itens estão corretos.

Gabarito: Letra E

---

18. (FGV/TRT 13/2022) As duas interfaces disponíveis no Framework Spring 5 que oferecem métodos para transformar objetos Java em XML e vice-versa são

a) Biding e Unbiding.
b) CharsetEncoder e CharsetDecoder.
c) Marshaller e Unmarshaller.
d) Serializable e Deserializable.
e) Stub e Skeleton.

Comentários:

Para transformarmos objetos Java em XML, e vice-versa, usamos as interfaces Marshaller e Unmarsheller. Essa interface fornece um objeto Marsheller a partir do uso do JAXBContext.

Gabarito: Letra C

---

19. (CEBRASPE/TRT 8/2022) Assinale a opção que apresenta a anotação que define no framework Spring uma classe como pertencente à camada de persistência. A

a) @Repository
b) @Component
c) @Service
d) @Transient
e) @Autowired

Comentários:

Vamos analisar as anotações trazidas pela questão:

- @Repository: permite que classes possam acessar o banco de dados de forma direta, fazendo com que a classe exerça o papel de um repositório de acesso aos dados;
- @Component: usado para indicar um componente, marcando a classe como *bean* para ser incluído na aplicação;
- @Service: marca uma classe responsável por chamar APIs ou efetuar algum outro tipo de serviço;
- @Transient: é uma anotação do JPA, usada para marcar um atributo como pertencente à fase transiente do ciclo de vida, isso é, a classe não será persistida;
- @Autowired: marca uma classe para a injeção de dependências;

Portanto, para marcarmos uma classe como pertencente à camada de persistência, usamos o @Repository.

Gabarito: Letra A

---

20. (CEBRASPE/PGDF/2021) Julgue o item seguinte, a respeito de JMS (Java Message Service), JUnit e Spring Framework.

O Spring WebFlux é compatível com Java 8 lambdas e Kotlin e tem a vantagem de permitir a criação de microsserviços com requisitos menos complexos.

Comentários:

O Spring WebFlux foi projetado com foco em programação reativa, que se beneficia muito da compatibilidade com lambdas do Java 8 e com a concisão oferecida por linguagens como Kotlin. Além disso, o WebFlux é leve e não bloqueante, o que o torna ideal para cenários como microsserviços e sistemas distribuídos. Ele não depende do stack tradicional de Servlets (como no Spring MVC) e pode operar com servidores reativos como Netty, que têm requisitos menos complexos em termos de threads e recursos, especialmente em arquiteturas de alta escalabilidade e muitas conexões simultâneas.

Gabarito: Certo

LISTA DE QUESTÕES

01. (INQC/CPTRANS/2024) Utilizando o framework Spring MVC, caso se deseje criar um controller, a anotação a ser utilizada no código é:

a) #Controller
b) &Controller
c) @Controller
d) %Controller

02. (FGV/TJ AP/2024) Breno está criando um serviço REST, que será disponibilizado por meio de um aplicativo Spring Boot. Como ele já conhece o padrão do REST, criou um método de inclusão no controlador, tendo como parâmetro uma entidade do tipo gerenciado pelo serviço.

Para que o parâmetro receba corretamente os dados fornecidos pela requisição, no formato JSON, Breno irá utilizar nesse parâmetro a anotação:

a) PostMapping;
b) RestController;
c) GetMapping;
d) RequestBody;
e) PathParam.

03. (FGV/TJ AP/2024) A analista Joelma está desenvolvendo o web service tjapRest utilizando o Spring Boot. Determinadas operações de tjapRest devem ser executadas assincronamente. Para implementar as operações assíncronas de forma simples e direta, Joelma recorreu a duas anotações padrões do Spring. A primeira anotação habilita o suporte do Spring à execução de métodos assíncronos. A segunda anotação marca determinado método como um candidato à execução assíncrona.

Joelma recorreu às anotações do Spring:

a) @EnableAsync e @Async;
b) @AsyncConfigurer e @Async;
c) @EnableAsync e @Asynchronous;
d) @AsyncConfigurer e @EnableAsync;
e) @AsyncConfigurer e @Asynchronous.

04. (FGV/TJ MS/2024) No framework Spring MVC, o tipo de bean especial no WebApplicationContext que tem o objetivo de auxiliar o DispatcherServlet a invocar um manipulador mapeado para uma solicitação é o:

a) FlashMapManager;
b) HandlerAdapter;
c) HandlerMapping;
d) LocaleResolver;
e) MultipartResolver.

05. (Instituto Verbena/TJ AC/2024) O Spring MVC (Model-View-Controller) é um framework do Spring que facilita o desenvolvimento de aplicativos da web baseados no padrão de arquitetura MVC. Ele fornece um modelo de programação flexível e robusto para criar aplicativos da web, permitindo a separação clara de responsabilidades entre o modelo de dados (Model), a lógica de apresentação (View) e o controle de fluxo (Controller). Qual anotação no Spring MVC é usada para extrair valores de caminho (partes da URL) e passá-los para um método do controlador como argumentos?

a) @RequestHeader
b) @RequestParam
c) @ResponseBody
d) @PathVariable

06. (COMPERVE/UFRN/2023) O Spring framework é uma ferramenta amplamente utilizada no desenvolvimento de aplicações Java Web. No ciclo de vida de um Spring Bean, é possível utilizar anotações em métodos que vão adicionar algum comportamento nos momentos de criação e na destruição desse Bean. As duas anotações utilizadas no spring para usar esses métodos customizados, no ciclo de vida dos Beans, são

a) @PostConstruct e @PreDestroy.
b) @PostConstruct e @BeforeDestroy.
c) @AfterConstruct e @PreDestroy.
d) @AfterConstruct e BeforeDestroy.

07. (COMPERVE/UFRN/2023) No Spring framework, o escopo de um Bean define a sua visibilidade e o seu ciclo de vida. Sobre os escopos do Spring Framework, analise as afirmativas abaixo, num contexto de uma aplicação web.

I Se nenhum escopo for especificado, o escopo padrão utilizado é o Application.
II O escopo Session retorna uma instância do Bean única sempre que for requerido.
III O escopo Prototype retorna uma instância diferente do Bean sempre que for requerido.
IV Se nenhum escopo for especificado, o escopo padrão utilizado é o Singleton.

Entre as afirmativas, estão corretas

a) III e IV.
b) I e II.
c) I e III.
d) II e IV.

08. (CEBRASPE/TBG/2023)

```
package com.example.springboot;

import
org.springframework.web.bind.annotation.GetMappi
ng;
import
org.springframework.web.bind.annotation.RestCont
```

```
roller;

@RestController
public class HelloController {
    @GetMapping("/")
     public String index() {
         return "Greetings from Spring Boot!";
     }
}
```

A partir do código precedente, julgue o item subsecutivo, relativo a Spring.

No Spring, a RestController é usada para marcar a classe como um controlador em que cada método retorna um objeto de domínio em vez de uma exibição.

09. (PR4/UFRJ/2023) A injeção de dependência é uma técnica de design usada para obter a inversão de controle. O Spring Framework oferece um recurso de injeção de dependência que permite aos objetos definir suas próprias dependências que o contêiner Spring posteriormente injeta nelas. Assinale a alternativa que NÃO faz parte dos recursos de injeção de dependência do Spring mais recente.

a) @Autowired
b) @Ioc
c) @Qualifier
d) @DependsOn
e) @Inject

10. (FCC/TRT 21/2023) O RestTemplate é uma classe do Spring Framework usada para fazer chamadas HTTP a serviços externos. A anotação usada para modificar o comportamento padrão do RestTemplate indicando que ele deve ser configurado para suportar o balanceamento de carga ao chamar serviços registrados em um servidor de descoberta, como o Eureka, é a anotação

a) @LoadBalanced
b) @LoadBalancedRestfull
c) @ReatfullLoadBalanced
d) @Component Balanced
e) @BeanLoadBalanced

11. (CEBRASPE/DP DF/2022) Julgue o próximo item, a respeito de Spring Framework, JDBC, iText, Java 8 e Apache CXF.

No Spring Framework, a anotação @RequestClass é usada para que a aplicação saiba a requisição que deve ser tratada pelo @Controller.

12. (VUNESP/ALESP/2022) Na Linguagem de Expressão do Spring Framework (Spring Expression Language, SpEL), uma String literal é delimitada por

a) aspas duplas.
b) sinais de menor e maior.
c) parênteses.
d) chaves.
e) aspas simples.

13. (FGV/TJ TO/2022) O técnico em informática Marcos implementou o web service REST obtemPong utilizando Java com framework Spring. O web service obtemPong recebe o parâmetro obrigatório ping.

Observe abaixo o principal trecho do código-fonte de obtemPong:

```
@GetMapping("/api/v1/pong") @ResponseBody
public String obtemPong(@RequestParam String ping) {
                return ping;
}
```

Para que o parâmetro ping deixe de ser obrigatório e automaticamente assuma o valor “pong” caso esteja ausente da mensagem de requisição, Marcos deve adicionar à anotação @RequestParam do parâmetro ping o argumento:

a) value = “pong”;
b) name = “pong”;
c) defaultValue = “pong”;
d) required = “ping?ping:pong”;
e) value = “ping?ping:pong”.

14. (FCC/TRT 19/2022) Na abordagem do Spring para construir serviços web RESTful, as solicitações HTTP são tratadas por um controlador, que é uma classe identificada com a anotação

a) @RestfulController
b) @RestfulMapping
c) @RestApplicationController
d) @RestController
e) @RequestController

15. (FCC/TRT 19/2022) Na abordagem do Spring para construir serviços web RESTful, as solicitações HTTP são tratadas por uma classe conhecida como controlador. Nessa classe, a anotação que garante que as requisições HTTP GET feitas para /dados sejam mapeadas para o método dados() é a anotação

a) @GetMappings("method=dados")
b) @RequestMappingMethod("/dados")
c) @GetMapping("/dados")
d) @GetMappingMethod("/dados")
e) @RequestMappings(method=dados)

16. (FCC/TRT 23/2022) A base do contêiner Inversion of Control (IoC), também conhecido como Dependency Injection (DI), do Spring Framework, é formada pelos pacotes

a) org.springframework.beans e org.springframework.context
b) org.springframework.orm e org.springframework.jdbc
c) org.springframework.web e org.springframework.webmvc
d) org.springframework.core e org.springframework.expression
e) org.springframework.webmvc e org.springframework.websocket

17. (IDECAN/TJ PI/2022) O Spring é um framework desenvolvido para a plataforma Java que facilita a vida do desenvolvedor quando falamos da construção de código de infraestrutura. Baseado na ideia da inversão de controle e injeção de dependência, Spring conta com diversos módulos que auxiliam na construção de aplicações corporativas.

A respeito dos conceitos e módulos presentes no framework, analise as afirmativas abaixo e marque alternativa correta.

I. No Spring a utilização da inversão de controle é facilitada graças à injeção de dependência.
II. @Autowired é a notação utilizada em Spring quando desejamos trabalhar com injeção de dependência por campo.
III. Spring Boot é um dos integrantes do framework do Spring. Tem foco na missão de facilitar o processo de configuração das aplicações. Essa facilitação ocorre graças ao conceito de convenção sobre a configuração.

a) Apenas as afirmativas I e II estão corretas.
b) Apenas as afirmativas I e III estão corretas.
c) Apenas a afirmativa I está correta.
d) Apenas as afirmativas II e III estão corretas.
e) Todas as afirmativas estão corretas.

18. (FGV/TRT 13/2022) As duas interfaces disponíveis no Framework Spring 5 que oferecem métodos para transformar objetos Java em XML e vice-versa são

a) Biding e Unbiding.
b) CharsetEncoder e CharsetDecoder.
c) Marshaller e Unmarshaller.
d) Serializable e Deserializable.
e) Stub e Skeleton.

19. (CEBRASPE/TRT 8/2022) Assinale a opção que apresenta a anotação que define no framework Spring uma classe como pertencente à camada de persistência. A

a) @Repository
b) @Component
c) @Service
d) @Transient
e) @Autowired

20. (CEBRASPE/PGDF/2021) Julgue o item seguinte, a respeito de JMS (Java Message Service), JUnit e Spring Framework.

O Spring WebFlux é compatível com Java 8 lambdas e Kotlin e tem a vantagem de permitir a criação de microsserviços com requisitos menos complexos.

GABARITO

1. Letra C
2. Letra D
3. Letra A
4. Letra B
5. Letra D
6. Letra A
7. Letra A
8. Certo
9. Letra B
10. Letra A
11. Errado
12. Letra E
13. Letra C
14. Letra D
15. Letra C
16. Letra A
17. Letra E
18. Letra C
19. Letra A
20. Certo

SPRING BOOT

Conceitos Gerais

O **Spring Boot** surgiu com o propósito de tornar ainda mais ágil o desenvolvimento de aplicações no ecossistema Spring Framework, especialmente ao **atenuar a necessidade de configurações extensas e repetitivas**. Embora o Spring já ofereça notável flexibilidade e modularidade, sua configuração detalhada — seja por meio de XML ou anotações em Java — podia se tornar trabalhosa para cada novo projeto, sobretudo em equipes que desejam entregar funcionalidades em prazos mais curtos.

Para mitigar essas dificuldades, o **Spring Boot adota um paradigma denominado opinionated ("orientado a convenções")**, no qual ele **assume escolhas-padrão e inclui automaticamente dependências relevantes**. Tal postura agiliza a criação de aplicações ao fornecer um ambiente pré-configurado de modo que, a partir do momento em que o projeto é gerado, um conjunto essencial de bibliotecas e configurações encontra-se pronto para uso. Contudo, caso seja necessário um nível mais profundo de personalização, o Spring Boot também permite a sobrescrita pontual dessas convenções.

SPRING BOOT → FACILITAR CONFIGURAÇÕES DE PROJETOS

Na prática, essa abordagem viabiliza a criação célere de aplicações Java, incluindo serviços web, pois o framework disponibiliza um servidor embutido (normalmente o Tomcat) e uma estrutura inicial que pode ser executada imediatamente. Isso reduz consideravelmente a complexidade de início de projeto, dispensando configurações extensivas e a instalação manual de servidores de aplicação. Ao mesmo tempo, para quem já interagiu com o Spring em sua forma clássica, há a vantagem de todo o poder e flexibilidade do ecossistema continuarem disponíveis para adaptações mais avançadas.

(IDECAN/TJ PI/2022 – *Adaptada*) A respeito dos conceitos e módulos presentes no framework Spring, julgue a alternativa subsecutiva.

Spring Boot é um dos integrantes do framework do Spring. Tem foco na missão de facilitar o processo de configuração das aplicações. Essa facilitação ocorre graças ao conceito de convenção sobre a configuração.

Comentários:

Perfeito! O Boot é um dos integrantes do ecossistema Spring, focando na parte de configuração de aplicações, simplificando e facilitando o processo a partir de uma abordagem orientada a convenções. *(Gabarito: Certo)*

Arquitetura

A arquitetura do Spring Boot é fundamentada no objetivo de simplificar o desenvolvimento de aplicações ao adotar convenções e automações que minimizam a necessidade de configurações explícitas, sem comprometer a flexibilidade. O Spring Boot é construído sobre o Spring Framework, utilizando os mesmos princípios de inversão de controle (IoC) e injeção de dependências, mas com um foco adicional em conveniência e produtividade.

A auto-configuração é o núcleo do Spring Boot. Quando a aplicação é iniciada, o Spring Boot analisa o classpath, verifica as dependências disponíveis e configura automaticamente os beans e serviços necessários. Por exemplo, se um banco de dados está disponível no classpath, o Spring Boot configura automaticamente uma fonte de dados (DataSource). Para personalizações, esse comportamento pode ser sobrescrito, permitindo configurações personalizadas por meio de arquivos de propriedades ou configurações explícitas em classes anotadas.

Os arquivos application.properties e application.yml são os principais meios de configuração da aplicação. Eles permitem configurar, de forma centralizada e declarativa, uma ampla gama de propriedades, como definições de servidor, banco de dados, segurança, cache, perfis e comportamento de módulos específicos do Spring ou de bibliotecas externas. Além disso, o Spring Boot permite usar arquivos específicos para cada perfil da aplicação, como application-dev.properties ou application-prod.yml.

Lembrando que o Spring Boot segue um padrão orientado a convenções. Então, ao iniciar uma aplicação, teremos algumas configurações embutidas – que podem ser substituídas nos arquivos de configuração. Por exemplo, um servidor embutido padrão é o Tomcat, enquanto para bancos de dados em memória, usamos o H2 para projetos com JPA.

Starters

Starters são **dependências predefinidas** que agrupam as bibliotecas e configurações básicas necessárias para implementar funcionalidades específicas em uma aplicação. Eles eliminam a necessidade de gerenciar manualmente várias dependências relacionadas a um mesmo propósito, proporcionando um início rápido e bem estruturado para projetos.

Um starter é, essencialmente, um pom.xml (Maven) ou um build.gradle com dependências que estão frequentemente associadas a determinada funcionalidade. Por exemplo, spring-boot-starter-web inclui bibliotecas para construir aplicações web usando Spring MVC, enquanto spring-boot-starter-data-jpa inclui dependências para JPA e Hibernate.

Esses startersvêm com configurações padrão, o que permite que o desenvolvedor comece rapidamente sem se preocupar com detalhes de configuração inicial. E como os starters centralizam dependências relacionadas, atualizar uma funcionalidade específica torna-se mais simples, pois basta alterar a versão do starter no arquivo de build.

Os principais *starters* são:

Pacote	Descrição
spring-boot-starter	O starter básico, inclui dependências essenciais como Spring Core, IoC e Logging (SLF4J, Logback).
spring-boot-starter-web	Inclui dependências para construir aplicações web (Spring MVC, Tomcat, Jackson para JSON, etc.).
spring-boot-starter-data-jpa	Inclui JPA, Hibernate e ferramentas para integração com bancos de dados relacionais.
spring-boot-starter-security	Fornece integração com o módulo Spring Security para autenticação e autorização.
spring-boot-starter-test	Inclui bibliotecas de teste (JUnit, Mockito, AssertJ, etc.) para testes de unidade e integração.
spring-boot-starter-actuator	Adiciona endpoints de monitoramento e gerenciamento para a aplicação.
spring-boot-starter-webflux	Inclui dependências para construir aplicações reativas com Spring WebFlux.
spring-boot-starter-validation	Fornece suporte a validações de dados, como Bean Validation (JSR-303).
spring-boot-starter-mail	Configura bibliotecas para envio de e-mails (JavaMail).

spring-boot-starter-logging	Configura o suporte a logging usando SLF4J e Logback.
spring-boot-starter-cache	Inclui suporte a caching para melhorar o desempenho da aplicação.

Por exemplo, a aplicação de uma dessas dependências pode ser feita da seguinte forma:

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Isso automaticamente inclui dependências para:

- Spring MVC.
- Servidor Tomcat embutido.
- Jackson para manipulação de JSON.
- Bibliotecas de logging (SLF4J e Logback).

(FCC/TRT 22/2022) À classe principal da aplicação Spring Boot, um Técnico adicionou a anotação @EnableEurekaServer para fazer com que a aplicação atue como um servidor Eureka (Discovery Server). Em seguida, adicionou ao arquivo de configuração Maven pom.xml uma dependência, como mostrado abaixo.

<dependency>
..I..
</dependency>

Para que a dependência adicionada seja do servidor Spring Cloud Eureka, a lacuna I deve ser corretamente preenchida por

(A) <packageName>org.springframework.cloud</packageName> <resourceName>
spring-cloud-eureka-server</resourceName>
(B) <packageName>org.springframework.cloud</packageName>
<resourceName>spring-cloud-dependencies</resourceName>
(C) <groupId>spring.cloud.framework</groupId>
<artifactId>eureka-discovery-server</artifactId>
(D) compile('org.springframework.cloud:spring-cloud-starter-eureka-server')
(E) <groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-eureka-server</artifactId>

Comentários:

Mesmo que não saibamos como é o endereço para incluirmos a dependência do Eureka, sabemos o seguinte:

- A sintaxe deve ser composta por um <groupId> e um <artifactId>, o que já elimina as letra A, B e D.
- O endereço para a dependência do Spring Framework é org.springframework.<componente>, no caso, como estamos trabalhando com o Spring Cloud, seria org.springframework.cloud.

Isso nos permite inferir que a resposta é a letra E. *(Gabarito: Letra E)*

Actuator

O Spring Boot Actuator é um módulo do Spring Boot que fornece funcionalidades para monitoramento, gerenciamento e diagnóstico de aplicações. Ele disponibiliza uma série de endpoints que permitem acessar informações sobre o estado interno da aplicação, como saúde do sistema, métricas de desempenho, configurações ativas, logs e beans gerenciados. O Actuator tem ganhado destaque, principalmente num contexto de DevOps, pois facilita o monitoramento em tempo real e a integração com ferramentas externas de monitoramento, como Grafana e Prometheus.

Ao incluir a dependência do Actuator, ele configura automaticamente os endpoints básicos com base no classpath e nas necessidades da aplicação. Porém, é possível customizar o comportamento dos endpoints por meio de configurações em application.properties ou application.yml.

Os *endpoints* prontos incluídos no Actuator incluem:

- /actuator/health: Indica o estado da aplicação (geralmente "UP" ou "DOWN"), podendo incluir verificações específicas, como banco de dados, filas de mensagens e conexões externas.
- /actuator/metrics: Exibe métricas detalhadas sobre a aplicação, como uso de memória, contagem de threads, requisições HTTP e tempo de resposta.
- /actuator/info: Exibe informações gerais da aplicação, como versão, descrição e metadados personalizados.
- /actuator/env: Lista as propriedades de configuração e variáveis de ambiente ativas.
- /actuator/loggers: Permite gerenciar dinamicamente o nível de logs para diferentes pacotes ou classes.
- /actuator/beans: Mostra todos os beans carregados no contexto do Spring, facilitando a inspeção.

Para habilitar o Actuator, usamos a seguinte dependência no pom.xml:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Endpoints

A criação de **endpoints Web e REST** no Spring Boot é simples e direta, graças à configuração automática proporcionada pelo framework. Por meio de anotações específicas, é possível mapear URLs para métodos Java, permitindo que a aplicação atenda requisições HTTP de forma eficiente e intuitiva. Essa funcionalidade é oferecida pelo Spring Web, que é configurado automaticamente ao incluir o starter `spring-boot-starter-web`.

Um endpoint no Spring Boot é normalmente implementado em uma classe anotada com **`@RestController`** ou **`@Controller`**, que define os métodos correspondentes às requisições HTTP. A anotação `@RestController` é especialmente útil para criar APIs RESTful, pois combina o comportamento de `@Controller` e `@ResponseBody`, retornando dados diretamente no corpo da resposta HTTP.

Veja um exemplo:

```java
@RestController
@RequestMapping("/api/v1")
public class MeuController {

    @GetMapping("/saudacao")
    public String saudacao() {
        return "Olá, bem-vindo ao Spring Boot!";
    }
}
```

No exemplo, temos:

- @RestController: Indica que a classe é um controlador REST.
- @RequestMapping("/api/v1"): Define o caminho base para todos os endpoints da classe.
- @GetMapping("/saudacao"): Mapeia requisições HTTP do tipo GET para o método saudacao.

Ao acessar http://localhost:8080/api/v1/saudacao, o cliente receberá a resposta "Olá, bem-vindo ao Spring Boot!".

Para mapeamentos, usamos o conjunto de anotações do `@RequestMapping` e seus variantes – como o `@GetMapping` para requisições GET, `@PostMapping` para POST, `@PutMapping` para PUT, e assim por diante. Além disso, podemos usar @RequestParam para extrair parâmetros de uma *query string*, ou `@PathVariable` para extrair valores dinâmicos de URLs.

Persistência e Data Access

O Spring Boot simplifica o acesso a dados e a persistência, fornecendo ferramentas que integram e abstraem diferentes tecnologias de banco de dados e frameworks de mapeamento objeto-relacional (ORM). Com sua configuração automática e o uso de starters, o Spring Boot permite que desenvolvedores criem aplicações que interajam com bancos de dados de forma rápida, eficiente e com menor esforço manual.

As integrações do Spring Boot incluem:

- Spring Data: Um conjunto de projetos do Spring que facilita a integração com bancos de dados relacionais e NoSQL. Inclui suporte a tecnologias como JPA, JDBC, MongoDB, entre outras.
- Spring Data JPA: Uma extensão do Spring Data para uso com JPA (Java Persistence API). Simplifica o mapeamento objeto-relacional, reduzindo o esforço de escrita de queries e interações com bancos de dados relacionais.
- Spring Data JDBC: Oferece uma abordagem mais direta para acesso a dados, sem o peso e a complexidade do JPA, sendo ideal para cenários em que o controle do SQL é mais importante.
- Banco de Dados NoSQL: Inclui suporte nativo a bancos NoSQL como MongoDB, Cassandra e Redis.
- H2 Database: Um banco de dados relacional em memória que é configurado automaticamente, ideal para desenvolvimento e testes.

A configuração é feita no pom.xml, definindo um starter com as dependências. Por exemplo, para definir um *starter* para JPA usamos:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

Atenção para a definição do JDBC – ela não é [...]-data-<componente>, como com as outras integrações:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
```

A anotação mais importante aqui é a `@Transactional`. Ela permite definir de forma declarativa como as transações devem ser iniciadas, gerenciadas e encerradas (com commit ou rollback), simplificando o controle sobre essas operações.

Integração com Contêineres

O Spring Boot introduziu **suporte nativo para a criação de camadas** (layers) em artefatos JAR e WAR, projetadas para facilitar a criação de imagens de contêineres, assim facilitando a integração com ferramentas como o Docker. A separação em camadas aproveita a capacidade do Docker de reutilizar partes inalteradas de uma imagem, resultando em imagens menores e builds mais rápidos, especialmente em aplicações frequentemente atualizadas.

O Spring Boot **organiza os componentes do JAR ou WAR em camadas baseadas na probabilidade de alteração entre builds**. Isso é feito para que partes estáveis (como dependências externas) permaneçam em camadas inferiores, enquanto o código da aplicação, que muda frequentemente, é isolado em uma camada superior. Esse isolamento minimiza o número de camadas reconstruídas a cada alteração.

Ao criar um artefato no Spring Boot, ele é automaticamente separado em camadas:

- **Camada de dependências:** Inclui bibliotecas externas (jars do classpath, como dependências Maven/Gradle). Essas dependências raramente mudam entre builds, sendo armazenadas em uma camada estável e reutilizável.
- **Camada de recursos do Spring Boot:** Contém arquivos internos ao Spring Boot (como classes de inicialização e configurações padrão). Geralmente, também não muda frequentemente, tornando-se reaproveitável.
- **Camada de código da aplicação:** Contém o código do desenvolvedor, incluindo controladores, serviços, repositórios, e quaisquer outras classes criadas especificamente para o projeto. Esta camada muda frequentemente, pois é onde ocorrem as alterações no desenvolvimento.
- **Camada de recursos estáticos e configurações:** Inclui arquivos como application.properties, application.yml e recursos estáticos (imagens, CSS, JS). Pode mudar dependendo da personalização ou ajustes de configuração.

Ao construir o artefato (JAR ou WAR) usando Maven ou Gradle, o Spring Boot gera automaticamente um arquivo chamado layers.idx. Esse arquivo contém a definição das camadas, listando quais arquivos pertencem a cada camada. Com isso, ferramentas como o Docker podem usar os Buildpacks, que utilizam o arquivo layers.idx para criar as imagens baseadas em camadas otimizadas.

Spring Boot CLI

O Spring Boot suporta uma série de comandos CLI. Aqui vocês já sabem, não tem muita explicação, temos que saber os principais comandos:

Comando	Descrição
java -jar <arquivo>.jar	Executa uma aplicação Spring Boot empacotada como um fat JAR.
mvn spring-boot:run	Executa uma aplicação Spring Boot diretamente pelo Maven, sem necessidade de empacotamento.
java -Dspring.profiles.active=<profile> -jar <arquivo>.jar	Executa a aplicação Spring Boot com um perfil específico ativado.
java -Dserver.port=<porta> -jar <arquivo>.jar	Executa a aplicação definindo uma porta específica para o servidor embutido.
java -Dlogging.level.<pacote>=<level> -jar <arquivo>.jar	Ajusta o nível de log para um pacote ou classe específicos em tempo de execução.
spring init <nome-do-projeto>	Gera um novo projeto Spring Boot com configuração básica usando o Spring Initializr (CLI).
spring run <arquivo>.groovy	Executa diretamente um arquivo Groovy que utiliza o Spring Boot (via Spring Boot CLI).
spring install <plugin>	Instala um plugin adicional para o Spring Boot CLI.
curl -X POST <URL>/actuator/shutdown	Encerra graciosamente a aplicação Spring Boot se o Actuator estiver habilitado com este endpoint.
java -jar <arquivo>.jar --<propriedade>=<valor>	Define uma propriedade Spring em tempo de execução (equivalente a configurá-la em application.properties).
java -Xms<valor> -Xmx<valor> -jar <arquivo>.jar	Define o limite mínimo e máximo de memória para a JVM ao executar a aplicação.
spring shell	Inicia um terminal interativo para gerenciar e executar comandos do Spring CLI.
spring help	Exibe uma lista com todos os comandos disponíveis no Spring CLI.

(CEBRASPE/DATAPREV/2023) Acerca de Spring Boot, julgue o item a seguir.

O uso do comando *spring init* é capaz de criar um novo projeto.

Comentários:

Perfeito, o comando para iniciarmos um novo projeto é o *spring init*. *(Gabarito: Certo)*

Anotações

As anotações, assim como os comandos CLI, são carta marcada em provas. Abaixo, reúno os principais que você deve levar para a sua prova:

Anotação	Descrição
@SpringBootApplication	Combina @Configuration, @EnableAutoConfiguration e @ComponentScan. É a anotação principal para inicializar uma aplicação Spring Boot.
@EnableAutoConfiguration	Habilita a configuração automática do Spring Boot com base nas dependências do classpath.
@ComponentScan	Realiza a varredura de pacotes para encontrar classes anotadas com @Component, @Service, @Repository, etc.
@Configuration	Marca uma classe como fonte de definições de beans e configurações.
@RestController	Combina @Controller e @ResponseBody. Marca uma classe como um controlador RESTful.
@Controller	Marca uma classe como um controlador Spring MVC, usada principalmente para páginas baseadas em view.
@RequestMapping	Mapeia URLs para classes ou métodos controladores, aplicando a diversos métodos HTTP (GET, POST, etc.).
@GetMapping	Especifica mapeamento de requisições HTTP do tipo GET para um método controlador.
@PostMapping	Especifica mapeamento de requisições HTTP do tipo POST para um método controlador.
@PutMapping	Especifica mapeamento de requisições HTTP do tipo PUT para um método controlador.
@DeleteMapping	Especifica mapeamento de requisições HTTP do tipo DELETE para um método controlador.
@RequestParam	Mapeia parâmetros da query string para os parâmetros do método controlador.
@PathVariable	Mapeia variáveis de URL (path) para os parâmetros do método controlador.
@RequestBody	Converte o corpo da requisição HTTP (geralmente JSON ou XML) para um objeto Java.

@ResponseBody	Indica que o valor de retorno de um método será diretamente escrito no corpo da resposta HTTP.
@EnableScheduling	Habilita o suporte para tarefas agendadas com @Scheduled.
@Scheduled	Configura métodos para serem executados periodicamente (usando cron jobs, intervalos, etc.).
@EnableAsync	Habilita o suporte para execução assíncrona com a anotação @Async.
@Async	Define que um método será executado de forma assíncrona.
@EnableCaching	Habilita o suporte para caching, permitindo o uso de anotações como @Cacheable e @CacheEvict.
@Cacheable	Define que o resultado de um método deve ser armazenado em cache.
@CacheEvict	Remove entradas do cache após a execução de um método.
@Profile	Especifica que o bean só será carregado para perfis específicos (como dev, test, prod).
@ConditionalOnProperty	Condicional baseada em propriedades: só ativa o bean se determinada propriedade estiver presente ou com valor específico.
@ConditionalOnMissingBean	Só ativa o bean se nenhum outro bean do mesmo tipo estiver presente no contexto.
@ConditionalOnClass	Só ativa o bean se uma determinada classe estiver presente no classpath.
@ConditionalOnMissingClass	Só ativa o bean se uma determinada classe não estiver presente no classpath.
@SpringBootTest	Indica que a classe de teste deve carregar o contexto completo da aplicação Spring Boot.
@TestConfiguration	Define uma classe de configuração que será usada apenas para testes.
@MockBean	Define um mock para um bean no contexto do Spring durante os testes.
@SpyBean	Define um spy para um bean no contexto do Spring durante os testes.
@DataJpaTest	Configura o ambiente para testes focados na camada de persistência (JPA e banco de dados).

@WebMvcTest	Configura o ambiente para testes focados na camada web (controladores, validação, etc.).
@RestClientTest	Configura o ambiente para testes focados em clientes REST, como RestTemplate ou WebClient.
@EnableConfigurationProperties	Habilita a vinculação de classes com @ConfigurationProperties às configurações da aplicação.
@ConfigurationProperties	Permite mapear propriedades do arquivo application.properties ou application.yml para classes POJO.

QUESTÕES COMENTADAS

01. (FGV/DATAPREV/2024 – *Adaptada*) No desenvolvimento de software, os frameworks Spring, Spring Cloud, Spring Boot, Hibernate e JUnit desempenham papéis importantes na construção de aplicações modernas. Nesse contexto, julgue o item abaixo.

O Spring Boot é responsável por fornecer uma solução para desenvolvimento de microsserviços escaláveis.

Comentários:

O item está correto! O Spring Boot é focado em facilitar o desenvolvimento de aplicações, com configurações automáticas e dependências pré-configuradas, permitindo criar serviços autossuficientes e pronto para a produção mais rapidamente.

Gabarito: Certo

---

02. (INSTITUTO ACESSO/CM Manaus/2024) Spring Boot e Spring MVC são frameworks populares em Java para o desenvolvimento de aplicações web e microserviços. Enquanto Spring MVC facilita a criação de controladores e rotas, Spring Boot simplifica a configuração e o deployment da aplicação.

Qual das alternativas a seguir descreve corretamente uma característica do Spring Boot?

a) Spring Boot automatiza a configuração de dependências e setups iniciais, permitindo criar aplicações com menos configurações manuais.
b) Spring Boot elimina a necessidade de usar anotações como @Controller ou @RestController, simplificando a definição de controladores.
c) Spring Boot substitui o uso do Spring MVC, uma vez que ambos possuem funcionalidades idênticas para criação de APIs REST.
d) Spring Boot limita a criação de aplicações apenas para ambientes de microserviços, não sendo aplicável a monolitos.
e) Spring Boot permite configurar a aplicação com extensos arquivos XML para garantir flexibilidade na customização.

Comentários:

Vamos analisar cada alternativa.

a) Certo. É exatamente esse o objetivo do Spring Boot.
b) Errado. O Spring Boot não elimina a necessidade de anotações, ele as utiliza.

c) Errado. O Spring Boot usa o Spring MVC como base, complementando-o, não o substituindo.
d) Errado. Podemos ter tanto aplicativos monolíticos quanto microsserviços.
e) Errado. O Spring Boot otimiza o trabalho, por isso só precisamos introduzir os pacotes necessários que já estarão pré-configurados. Ainda, é possível configurar dependências e componentes pontuais, customizando o *boot*.

Portanto, correta a letra A.

Gabarito: Letra A

---

03. (FGV/TRF 1/2024) O analista Eric foi designado para compatibilizar um antigo projeto de software Java com o recente Spring Boot 3. A versão atual do projeto utiliza Spring Boot 2.1 com Java 8, além de algumas Application Programming Interfaces (APIs) do Java Enterprise Edition (JEE).

A fim de atualizar o projeto para o Spring Boot 3 observando estritamente o mínimo necessário, Eric deve atualizar o Java para a versão:

a) 11, mantendo o uso das APIs do Java EE;
b) 17, mantendo o uso das APIs do Java EE;
c) 21, mantendo o uso das APIs do Java EE;
d) 17, migrando o uso das APIs do Java EE para o Jakarta EE;
e) 21, migrando o uso das APIs do Java EE para o Jakarta EE.

Comentários:

Questão um pouco “leviana”. Mas vamos lá, para usarmos o Spring Boot 3, devemos ter ao menos Java 17, sendo compatível até a versão do Java 23. Além disso, o Boot migrou do uso das APIs do Java EE para o Jakarta EE.

Gabarito: Letra D

---

04. (FGV/TRF 1/2024) O analista Fábio precisa adicionar o suporte a um segundo DataSource, chamado DS2, em uma aplicação baseada em Spring Boot. Cada DataSource da aplicação é conectado a um banco de dados diferente. O projeto com o código da aplicação possui o arquivo de parâmetros do Spring Boot application.properties, que deve ser modificado por Fábio para acelerar a implementação.

Para adicionar o suporte ao segundo banco de dados no Spring Boot, Fábio deve adicionar ao application.properties os parâmetros do DS2 e:

a) ativar a property spring.datasource.multiple;
b) configurar a property spring.datasource.active=2;
c) ativar a property spring.datasource.allow-second;
d) declarar um bean PropertyMapper para cada DataSource;
e) declarar um bean ConfigurationProperties para cada DataSource.

Comentários:

No Spring Boot, o suporte para múltiplos DataSources não é configurado automaticamente. Quando a aplicação precisa se conectar a mais de um banco de dados, é necessário declarar explicitamente os DataSources adicionais como beans, utilizando a anotação @ConfigurationProperties para associá-los às respectivas configurações no arquivo application.properties.

Na prática, o arquivo application.properties deve incluir as propriedades para cada DataSource, geralmente nomeados de forma clara, como spring.datasource.primary e spring.datasource.secondary. Além disso, cada DataSource deve ser configurado como um bean, utilizando `@ConfigurationProperties` para vincular as propriedades às configurações declaradas no arquivo.

Gabarito: Letra E

---

05. (FCC/TRT 7/2024) Para responder a questão, considere a Arquitetura de desenvolvimento da Plataforma Digital do Poder Judiciário - PDPJ-Br.

Os microsserviços da PDPJ-Br são desenvolvidos com tecnologias de código aberto e são implementados com o uso do framework Spring. Dentre os recursos inicializados pelo Spring Boot está o Actuator Micrometer que é

a) um mecanismo de suporte avançado para a camada de acesso a dados baseada em JPA.
b) um paradigma de programação que visa aumentar a modularidade, permitindo a separação de interações transversais no código.
c) uma biblioteca de instrumentação de métricas para aplicativos Java. Coleta as informações da aplicação em tempo de execução e as expõe em um endpoint.
d) uma ferramenta para organização de scripts SOL que são executados no banco de dados, funcionando como um controle de versão dele.
e) um banco de dados de série temporal dimensional. Extrai métricas de instâncias de aplicativos periodicamente com base na descoberta de serviço.

Comentários:

Um Actuator é uma biblioteca de suporte ao Spring Boot, que coleta informações sobre a aplicação e as leva até endpoints, servindo para integração com sistemas de monitoramento e avaliação, como Prometheus e Grafana.

Gabarito: Letra C

---

06. (UFMT/PREF. CÁCERES/2024) No desenvolvimento de aplicações Spring Boot, qual módulo fornece funcionalidades de monitoramento e gestão da aplicação em tempo de execução?

a) Spring Boot Actuator
b) Spring Boot Starter
c) Spring Boot Web
d) Spring Boot AutoConfigurator

Comentários:

A ferramenta da suíte do Spring Boot responsável pelo monitoramento e gestão é o Spring Boot Actuator.

Gabarito: Letra A

---

07. (FGV/TJ AP/2024) Breno está criando um serviço REST, que será disponibilizado por meio de um aplicativo Spring Boot. Como ele já conhece o padrão do REST, criou um método de inclusão no controlador, tendo como parâmetro uma entidade do tipo gerenciado pelo serviço.

Para que o parâmetro receba corretamente os dados fornecidos pela requisição, no formato JSON, Breno irá utilizar nesse parâmetro a anotação:

a) PostMapping;
b) RestController;
c) GetMapping;
d) RequestBody;
e) PathParam.

Comentários:

Queremos uma anotação para receber os dados da requisição em JSON. Vamos analisar as alternativas:

a) Errado. PostMapping: Mapeia requisições HTTP do tipo POST para um método específico do controlador, indicando que aquele método será chamado quando uma requisição POST for recebida na rota configurada.

b) Errado. RestController: Marca a classe como um controlador REST, fazendo com que os métodos retornem dados (normalmente em JSON ou XML) diretamente no corpo da resposta HTTP, em vez de retornarem páginas ou views.
c) Errado. GetMapping: Mapeia requisições HTTP do tipo GET para um método específico do controlador, definido para lidar com leituras ou consultas de recursos na rota indicada.
d) Certo. RequestBody: Associa o conteúdo do corpo da requisição (geralmente em JSON ou XML) a um parâmetro do método, convertendo os dados recebidos em um objeto Java que o método pode processar.
e) Errado. PathParam: No contexto de JAX-RS (e não do Spring MVC), especifica que o valor de um parâmetro de método deve ser obtido diretamente de uma parte do caminho (path) da URL, permitindo rotas com variáveis dinâmicas.

Portanto, correta a letra D.

Gabarito: Letra D

---

08. (FGV/TCE SP/2023) O analista Jacó implementou a aplicação TCERestAPI utilizando Java com Spring Boot. A TCERestAPI é apta para o deploy em servidores de aplicação Java preexistentes, mas também suporta a execução standalone do Spring Boot.

Para viabilizar ambas as formas de execução da TCERestAPI, Jacó precisou modificar a classe principal da aplicação, fazendo com que ela estendesse diretamente determinada classe do Spring Boot.

Jacó fez com que a classe principal da TCERestAPI estendesse a classe do Spring Boot:

a) SpringBootStarterTomcat;
b) SpringBootServletInitializer;
c) SpringBootStarterUndertow;
d) ServletContextInitializerBeans;
e) SpringServletContainerInitializer.

Comentários:

Para que uma aplicação Spring Boot seja apta a rodar tanto como um aplicativo standalone (com um servidor embutido, como Tomcat ou Jetty) quanto ser implantada em um servidor de aplicação Java preexistente (como um WAR em um container Servlet), a classe principal **precisa estender SpringBootServletInitializer**. Essa classe permite integrar o Spring Boot com o ciclo de vida do contêiner de Servlets.

Gabarito: Letra B

---

09. (CEBRASPE/DATAPREV/2023) Acerca de Spring Boot, julgue o item a seguir.

Como pré-requisito para a execução de um spring boot, é necessário ter uma versão válida do Java instalado.

Comentários:

Cespe deu uma colher de chá nessa, né? Obviamente, como o Spring Boot é usado em Java, precisamos ter uma versão válida instalada.

Gabarito: Certo

---

10. (CEBRASPE/SERPRO/2023) Julgue o próximo item, relativo a tecnologias backend.

Para facilitar a criação de imagens otimizadas do Docker, o Spring Boot suporta a adição de um arquivo de índice de camada ao jar e também suporta camadas para arquivos war, projetadas para separar o código com base na probabilidade de alteração entre as compilações do aplicativo, uma vez que é mais provável que o código do aplicativo mude entre as compilações; logo, o código é isolado em uma camada separada.

Comentários:

O Spring Boot inclui suporte nativo para otimizar a criação de imagens Docker usando o conceito de camadas (layers) em seus artefatos JAR e WAR. Esse recurso é especialmente útil para separar o código da aplicação de suas dependências, maximizando o reaproveitamento de camadas no Docker entre compilações.

Gabarito: Certo

---

11. (CEBRASPE/Pref. Fortaleza/2023) Com relação aos conceitos de Spring Boot, Net Core e thread, julgue o item seguinte.

Spring Boot é uma tecnologia que se integra a ferramentas e linguagens de desenvolvimento web a fim de otimizar seu código, a partir de pequenos ajustes e trocas para deixar mais rápido o resultado do código.

Comentários:

O Spring Boot foca na facilitação do *boot* da aplicação, isso, é, sua inicialização e toda a parte de configurações iniciais, como importação e setagem de dependências. Ele não objetiva interagir com linguagens de desenvolvimento web – seu foco é 100% no Java e no Spring Framework.

Gabarito: Errado

---

12. (FCC/TRT 17/2022) Em uma aplicação que utiliza Spring Boot, em condições ideais, o arquivo index.html possui o formulário abaixo.

```
<form method="POST" action="login">
   <label for="usuario">Usuário:</label>
   <input type="text" name="user" id="usuario">
   <label for="senha">Senha:</label>
   <input type="password" name="senha" id="senha">
   <input type="submit" value="Entrar">
</form>
```

Na classe Controle.java dessa aplicação, que possui a anotação @Controller, há um método chamado receberLogin. Para indicar que este método deve receber os dados do formulário acima ao se clicar no botão Entrar, imediatamente após a declaração deste método deve ser colocada a anotação

a) @RequestMapping(value = "/login", method = RequestMethod.POST)
b) @Request(HttpMethod = "POST" param = "/login")
c) @HttpServletRequest(method = "POST" requestFrom = "/login")
d) @SpringMapping(method = "post", paramRequest = "/login")
e) @HttpServletRequest(value = "/login", method = RequestMethod.POST)

Comentários:

No Spring MVC, para mapear requisições HTTP a métodos controladores, utilizamos anotações como @RequestMapping (ou suas especializações). Essa anotação permite definir:

- O caminho da requisição (value).
- O método HTTP (method), como GET, POST, etc.

No caso apresentado, o formulário está enviando uma requisição POST para o endpoint /login, o que requer que o método no controlador seja mapeado para aceitar essa combinação de caminho e método HTTP. Portanto, usaremos @RequestMapping(value = "/login", method = RequestMethod.POST), pois reflete a sintaxe apropriada para mapear o endpoint /login com o método HTTP POST no Spring MVC.

Gabarito: Letra A

---

13. (FCC/TRT 5/2022) Em uma aplicação construída com Spring Boot, em condições ideais, há um arquivo index.html, onde consta o link abaixo.

```
<a href="validar"> Validar </a>
```

Ao clicar nesse link uma requisição é enviada para uma classe anotada com @RestController, onde consta o método seguinte.

```
public String getValida() {
   return "Validado";
}
```

Para que essa requisição seja direcionada corretamente ao método getValida(), esse método precisa estar anotado, imediatamente antes de sua declaração, com

a) @MappingRequest("/validar")
b) @ServletMapping(path="validar")
c) @Mapping(path="/validar")
d) @ServletMapping("/validar")
e) @RequestMapping("/validar")

Comentários:

No Spring Boot, métodos dentro de classes anotadas com @RestController precisam ser mapeados para um caminho de URL para que possam processar requisições HTTP. Isso é feito com a anotação @RequestMapping ou suas variações, como @GetMapping, @PostMapping, entre outras. Nesse sentido, a alternativa correta é a letra E, pois reflete a prática padrão do Spring Boot para mapear caminhos de requisições a métodos controladores.

Gabarito: Letra E

---

14. (FCC/TRT 22/2022) As versões mais recentes do Spring Boot permitem a configuração de inicialização lenta (lazy initialization), que faz com que os beans sejam criados à medida que são necessários e não durante a inicialização do aplicativo. Uma das maneiras de habilitar a inicialização lenta é colocar

a) @Lazy(true) no arquivo application.properties
b) spring.main.lazy-initialization=true no arquivo application.properties
c) set-lazy-initialization=true no arquivo spring-boot.xml
d) lazy-initialization=true no arquivo spring-application-builder.xml
e) @SpringMainLazyInitialization na classe SpringApplicationBuilder

Comentários:

As versões mais recentes do Spring Boot oferecem suporte à inicialização lenta (lazy initialization) como uma opção de otimização. Essa abordagem faz com que os beans sejam criados apenas quando realmente necessários, em vez de serem todos inicializados durante o início da aplicação (eager initialization). Isso pode reduzir o tempo de inicialização em alguns cenários, especialmente em ambientes de desenvolvimento ou em aplicações grandes. A forma recomendada de habilitar essa funcionalidade é por meio da propriedade `spring.main.lazy-initialization=true`, nos arquivos application.properties ou application.yml.

Gabarito: Letra B

---

15. (FCC/TRT 22/2022) Um Técnico deseja incluir as configurações abaixo para serem executadas pelo Spring Boot quando a aplicação for iniciada.

```
spring.application.name = spring-cloud-config-server
server.port=8888
spring.cloud.config.server.git.uri = file:///c:/Users/test/config-files
```

Estas configurações devem ser inseridas no arquivo

a) spring-boot.xml
b) spring-boot.properties
c) spring-application.xml
d) spring-server.properties
e) application.properties

Comentários:

No Spring Boot, o arquivo application.properties (ou sua versão YAML, application.yml) é o local padrão para armazenar configurações da aplicação. Ele é automaticamente reconhecido e carregado pelo framework durante a inicialização, permitindo configurar propriedades como nome da aplicação, porta do servidor, e outras configurações específicas de módulos ou bibliotecas. Pela sintaxe, percebemos se tratar do arquivo application.properties.

Gabarito: Letra E

QUESTÕES COMENTADAS

01. (FGV/DATAPREV/2024 – *Adaptada*) No desenvolvimento de software, os frameworks Spring, Spring Cloud, Spring Boot, Hibernate e JUnit desempenham papéis importantes na construção de aplicações modernas. Nesse contexto, julgue o item abaixo.

O Spring Boot é responsável por fornecer uma solução para desenvolvimento de microsserviços escaláveis.

02. (INSTITUTO ACESSO/CM Manaus/2024) Spring Boot e Spring MVC são frameworks populares em Java para o desenvolvimento de aplicações web e microserviços. Enquanto Spring MVC facilita a criação de controladores e rotas, Spring Boot simplifica a configuração e o deployment da aplicação.

Qual das alternativas a seguir descreve corretamente uma característica do Spring Boot?

a) Spring Boot automatiza a configuração de dependências e setups iniciais, permitindo criar aplicações com menos configurações manuais.
b) Spring Boot elimina a necessidade de usar anotações como @Controller ou @RestController, simplificando a definição de controladores.
c) Spring Boot substitui o uso do Spring MVC, uma vez que ambos possuem funcionalidades idênticas para criação de APIs REST.
d) Spring Boot limita a criação de aplicações apenas para ambientes de microserviços, não sendo aplicável a monolitos.
e) Spring Boot permite configurar a aplicação com extensos arquivos XML para garantir flexibilidade na customização.

03. (FGV/TRF 1/2024) O analista Eric foi designado para compatibilizar um antigo projeto de software Java com o recente Spring Boot 3. A versão atual do projeto utiliza Spring Boot 2.1 com Java 8, além de algumas Application Programming Interfaces (APIs) do Java Enterprise Edition (JEE).

A fim de atualizar o projeto para o Spring Boot 3 observando estritamente o mínimo necessário, Eric deve atualizar o Java para a versão:

a) 11, mantendo o uso das APIs do Java EE;
b) 17, mantendo o uso das APIs do Java EE;
c) 21, mantendo o uso das APIs do Java EE;
d) 17, migrando o uso das APIs do Java EE para o Jakarta EE;
e) 21, migrando o uso das APIs do Java EE para o Jakarta EE.

04. (FGV/TRF 1/2024) O analista Fábio precisa adicionar o suporte a um segundo DataSource, chamado DS2, em uma aplicação baseada em Spring Boot. Cada DataSource da aplicação é conectado a um banco de dados diferente. O projeto com o código da aplicação possui o arquivo de parâmetros do Spring Boot application.properties, que deve ser modificado por Fábio para acelerar a implementação.

Para adicionar o suporte ao segundo banco de dados no Spring Boot, Fábio deve adicionar ao application.properties os parâmetros do DS2 e:

a) ativar a property spring.datasource.multiple;
b) configurar a property spring.datasource.active=2;
c) ativar a property spring.datasource.allow-second;
d) declarar um bean PropertyMapper para cada DataSource;
e) declarar um bean ConfigurationProperties para cada DataSource.

05. (FCC/TRT 7/2024) Para responder a questão, considere a Arquitetura de desenvolvimento da Plataforma Digital do Poder Judiciário - PDPJ-Br.

Os microsserviços da PDPJ-Br são desenvolvidos com tecnologias de código aberto e são implementados com o uso do framework Spring. Dentre os recursos inicializados pelo Spring Boot está o Actuator Micrometer que é

a) um mecanismo de suporte avançado para a camada de acesso a dados baseada em JPA.
b) um paradigma de programação que visa aumentar a modularidade, permitindo a separação de interações transversais no código.
c) uma biblioteca de instrumentação de métricas para aplicativos Java. Coleta as informações da aplicação em tempo de execução e as expõe em um endpoint.
d) uma ferramenta para organização de scripts SOL que são executados no banco de dados, funcionando como um controle de versão dele.
e) um banco de dados de série temporal dimensional. Extrai métricas de instâncias de aplicativos periodicamente com base na descoberta de serviço.

06. (UFMT/PREF. CÁCERES/2024) No desenvolvimento de aplicações Spring Boot, qual módulo fornece funcionalidades de monitoramento e gestão da aplicação em tempo de execução?

a) Spring Boot Actuator
b) Spring Boot Starter
c) Spring Boot Web
d) Spring Boot AutoConfigurator

07. (FGV/TJ AP/2024) Breno está criando um serviço REST, que será disponibilizado por meio de um aplicativo Spring Boot. Como ele já conhece o padrão do REST, criou um método de inclusão no controlador, tendo como parâmetro uma entidade do tipo gerenciado pelo serviço.

Para que o parâmetro receba corretamente os dados fornecidos pela requisição, no formato JSON, Breno irá utilizar nesse parâmetro a anotação:

a) PostMapping;
b) RestController;
c) GetMapping;
d) RequestBody;
e) PathParam.

08. (FGV/TCE SP/2023) O analista Jacó implementou a aplicação TCERestAPI utilizando Java com Spring Boot. A TCERestAPI é apta para o deploy em servidores de aplicação Java preexistentes, mas também suporta a execução standalone do Spring Boot.

Para viabilizar ambas as formas de execução da TCERestAPI, Jacó precisou modificar a classe principal da aplicação, fazendo com que ela estendesse diretamente determinada classe do Spring Boot.

Jacó fez com que a classe principal da TCERestAPI estendesse a classe do Spring Boot:

a) SpringBootStarterTomcat;
b) SpringBootServletInitializer;
c) SpringBootStarterUndertow;
d) ServletContextInitializerBeans;
e) SpringServletContainerInitializer.

09. (CEBRASPE/DATAPREV/2023) Acerca de Spring Boot, julgue o item a seguir.

Como pré-requisito para a execução de um spring boot, é necessário ter uma versão válida do Java instalado.

10. (CEBRASPE/SERPRO/2023) Julgue o próximo item, relativo a tecnologias backend.

Para facilitar a criação de imagens otimizadas do Docker, o Spring Boot suporta a adição de um arquivo de índice de camada ao jar e também suporta camadas para arquivos war, projetadas para separar o código com base na probabilidade de alteração entre as compilações do aplicativo, uma vez que é mais provável que o código do aplicativo mude entre as compilações; logo, o código é isolado em uma camada separada.

11. (CEBRASPE/Pref. Fortaleza/2023) Com relação aos conceitos de Spring Boot, Net Core e thread, julgue o item seguinte.

Spring Boot é uma tecnologia que se integra a ferramentas e linguagens de desenvolvimento web a fim de otimizar seu código, a partir de pequenos ajustes e trocas para deixar mais rápido o resultado do código.

12. (FCC/TRT 17/2022) Em uma aplicação que utiliza Spring Boot, em condições ideais, o arquivo index.html possui o formulário abaixo.

```
<form method="POST" action="login">
   <label for="usuario">Usuário:</label>
   <input type="text" name="user" id="usuario">
   <label for="senha">Senha:</label>
   <input type="password" name="senha" id="senha">
   <input type="submit" value="Entrar">
</form>
```

Na classe Controle.java dessa aplicação, que possui a anotação @Controller, há um método chamado receberLogin. Para indicar que este método deve receber os dados do formulário acima ao se clicar no botão Entrar, imediatamente após a declaração deste método deve ser colocada a anotação

a) @RequestMapping(value = "/login", method = RequestMethod.POST)
b) @Request(HttpMethod = "POST" param = "/login")
c) @HttpServletRequest(method = "POST" requestFrom = "/login")
d) @SpringMapping(method = "post", paramRequest = "/login")
e) @HttpServletRequest(value = "/login", method = RequestMethod.POST)

13. (FCC/TRT 5/2022) Em uma aplicação construída com Spring Boot, em condições ideais, há um arquivo index.html, onde consta o link abaixo.

```
<a href="validar"> Validar </a>
```

Ao clicar nesse link uma requisição é enviada para uma classe anotada com @RestController, onde consta o método seguinte.

```
public String getValida() {
   return "Validado";
}
```

Para que essa requisição seja direcionada corretamente ao método getValida(), esse método precisa estar anotado, imediatamente antes de sua declaração, com

a) @MappingRequest("/validar")
b) @ServletMapping(path="validar")
c) @Mapping(path="/validar")
d) @ServletMapping("/validar")
e) @RequestMapping("/validar")

14. (FCC/TRT 22/2022) As versões mais recentes do Spring Boot permitem a configuração de inicialização lenta (lazy initialization), que faz com que os beans sejam criados à medida que são necessários e não durante a inicialização do aplicativo. Uma das maneiras de habilitar a inicialização lenta é colocar

a) @Lazy(true) no arquivo application.properties
b) spring.main.lazy-initialization=true no arquivo application.properties
c) set-lazy-initialization=true no arquivo spring-boot.xml
d) lazy-initialization=true no arquivo spring-application-builder.xml
e) @SpringMainLazyInitialization na classe SpringApplicationBuilder

15. (FCC/TRT 22/2022) Um Técnico deseja incluir as configurações abaixo para serem executadas pelo Spring Boot quando a aplicação for iniciada.

```
spring.application.name = spring-cloud-config-server
server.port=8888
spring.cloud.config.server.git.uri = file:///c:/Users/test/config-files
```

Estas configurações devem ser inseridas no arquivo

a) spring-boot.xml
b) spring-boot.properties
c) spring-application.xml
d) spring-server.properties
e) application.properties

GABARITO

GABARITO

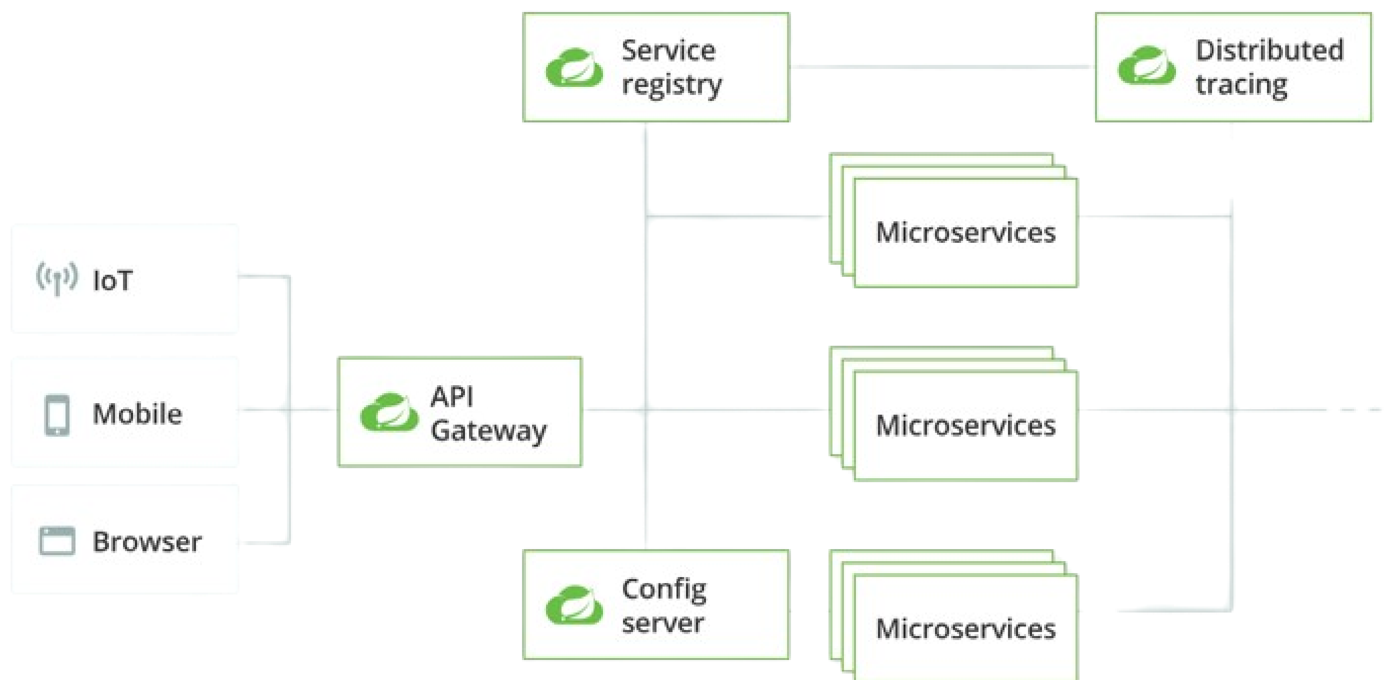
1. Certo
2. Letra A
3. Letra D
4. Letra E
5. Letra C
6. Letra A
7. Letra D
8. Letra B
9. Certo
10. Certo
11. Errado
12. Letra A
13. Letra E
14. Letra B
15. Letra E

SPRING CLOUD

Conceitos Gerais

O Spring Cloud é um componente dentro do ecossistema Spring que tem como principal objetivo **simplificar a construção de sistemas distribuídos**, especialmente aqueles baseados em microsserviços. O Spring Cloud surge como um conjunto de ferramentas para abordar essas arquiteturas. A ideia é fornecer recursos prontos para que o desenvolvedor não precise lidar manualmente com problemas clássicos de sistemas distribuídos. Ele integra-se muito bem ao Spring Boot, já que **ambos adotam a filosofia de "convenção sobre configuração"** e se beneficiam de mecanismos de autoconfiguração.

O Spring Cloud oferece uma game de soluções: Eureka, Gateway, Circuit Breaker, Sleuth, Stream, entre outros. Todas essas funcionalidades coexistem dentro do "guarda-chuva" do Spring Cloud, cada uma resolvendo um problema específico da arquitetura distribuída. O ponto comum é que elas partem dos princípios do Spring Boot: tudo é configurável via arquivos de propriedades (ou YAML), há mecanismos de auto-configuração e integração suave, e cada biblioteca pode ser adicionada através de starters (dependências que já trazem configurações pré-definidas).



Vamos lá!

Arquitetura

A arquitetura do Spring Cloud pode ser entendida como **um conjunto de módulos**, cada um voltado para resolver um aspecto específico da complexidade de sistemas distribuídos. Diferentemente de um único framework monolítico, o Spring Cloud adota uma abordagem de "caixa de ferramentas", onde cada módulo (ou subprojeto) atende uma necessidade particular, permitindo que os desenvolvedores combinem tais ferramentas de acordo com os requisitos de seus microsserviços.

De forma geral, podemos definir a arquitetura do Spring Cloud em três camadas principais:

- **Camada de Configuração e Gestão:** lida com o desafio de administrar parâmetros e configurações que podem variar entre ambientes (desenvolvimento, homologação, produção) e entre diferentes serviços. A funcionalidade mais conhecida aqui é o Spring Cloud Config, que fornece um servidor central de configuração, tipicamente apoiado em um repositório Git ou outro mecanismo de armazenamento. Todos os serviços do sistema acessam esse servidor para obter (ou atualizar) suas configurações. Essa arquitetura elimina a dispersão de propriedades e favorece a padronização, além de permitir mudanças em tempo de execução.
- **Camada de Descoberta, Roteamento e Comunicação:** nessa camada, encontramos soluções para permitir que serviços descubram uns aos outros, façam roteamento de requisições de forma inteligente e lidem com balanceamento de carga. Podemos destacar:
  - **Service Discovery (Eureka, Consul, Zookeeper):** Os microsserviços se registram em um servidor de descoberta e, ao mesmo tempo, consultam esse servidor para descobrir onde estão rodando os demais serviços.
  - **API Gateway (Spring Cloud Gateway):** Centraliza as requisições externas e internas, fornecendo recursos de roteamento, autenticação e autorização, e até manipulações específicas de rota (adicionar cabeçalhos, logar chamadas, etc.).
  - **Balanceamento de Carga (Spring Cloud Load Balancer):** Quando um serviço chama outro, o Spring Cloud pode integrar-se ao mecanismo de registro de serviços, espalhando as requisições entre as diversas instâncias disponíveis. Isso acontece sem necessidade de hardcode de endereços IP.
- **Camada de Resiliência, Observabilidade e Mensageria:** à medida que sistemas distribuídos se tornam mais complexos, é fundamental dotá-los de recursos que garantam robustez e forneçam visibilidade de seu comportamento. Nessa camada, encontramos:
  - **Circuit Breakers (Resilience4j, Hystrix):** Ajudam o serviço a lidar com falhas em chamadas remotas. Se um serviço dependente está offline ou lento, o circuito "abre" e impede novas chamadas momentaneamente, devolvendo respostas alternativas ou mensagens de erro sem prejudicar o sistema inteiro.
  - **Tracing Distribuído (Spring Cloud Sleuth, Zipkin):** Marca cada requisição com identificadores únicos para permitir rastrear sua jornada através de múltiplos microsserviços. Isso facilita a identificação de gargalos ou pontos de falha.
  - **Spring Cloud Stream:** Endereça a comunicação assíncrona (orientada a eventos) com o uso de mensagerias como Kafka ou RabbitMQ, permitindo arquiteturas "event-driven".
  - **Spring Cloud Bus:** Possibilita a propagação de mensagens de evento ou alterações de configuração entre diferentes instâncias de serviços.

Nosso estudo do Spring Cloud seguirá essas camadas.

Camada de Configuração e Gestão

A **camada de configuração e gestão** no Spring Cloud concentra as soluções voltadas para armazenar, carregar e gerenciar parâmetros essenciais dos microsserviços de forma centralizada. Em um sistema distribuído, cada serviço pode ter um conjunto distinto de propriedades (endereço de banco de dados, credenciais, chaves de API, URLs de serviços externos) e elas podem mudar conforme o ambiente (desenvolvimento, homologação, produção). A partir de certo porte de aplicação, manter todas essas configurações diretamente nos arquivos **`application.properties`** de cada serviço tende a gerar confusões e divergências, além de tornar tedioso atualizar cada instância separadamente.

O **Spring Cloud Config** é o principal componente dessa camada. Ele oferece um **Config Server**, que atua como ponto único para gerenciamento das configurações de todos os microsserviços. O Config Server carrega o "Serviço de Configuração", armazenando os arquivos de configuração num repositório Git, um repositório SVN ou o próprio sistema de arquivos local. Cada conjunto de propriedades é normalmente versionado em um arquivo por perfil e serviço, como `application-dev.properties`, `application-prod.yml` ou `usuario-service.yml`.

SPRING CLOUD CONFIG → CONFIGURAÇÕES GLOBAIS E DE PERFIL

Temos os ***Config Clients***, que são basicamente qualquer aplicação que precise obter suas configurações do Config Server. Para isso, basta incluir a dependência **`spring-cloud-starter-config`** e definir, no `application.properties`, as informações de localização do Config Server. Assim, ao iniciar a sua execução, o serviço cliente requisita ao Config Server as propriedades compatíveis com seu "application name" (ou seja, o nome do serviço) e o perfil ativo (dev, prod etc.), recebendo todas as configurações de maneira centralizada.

Em projetos reais, cada microsserviço pode precisar de configurações específicas para o perfil dev, test, prod ou outros. O Spring Cloud permite mapear perfis adicionais por meio de arquivos como `application-dev.properties`, `application-test.yml` etc. O conceito de "label" (ou rótulo) mapeia-se normalmente a branches (ou tags) no Git. Dessa forma, é possível ter uma branch main para produção e outra dev para desenvolvimento. O Config Server consegue servir configurações de acordo com o label solicitado pelos serviços.

É interessante notar que, antes de carregar suas propriedades específicas, o serviço cliente carrega as informações que apontam para o Config Server (por exemplo, `spring.cloud.config.uri`). Essas informações fazem parte de um "bootstrap" inicial, podendo inclusive ficar em um arquivo `bootstrap.properties`.

Camada de Descoberta, Roteamento e Comunicação

A **camada de descoberta, roteamento e comunicação** é a parte responsável por **garantir que os diversos microsserviços em um ambiente distribuído possam se localizar, trocar informações e encaminhar requisições entre si de maneira dinâmica e flexível**.

Em sistemas monolíticos, a comunicação interna é direta, pois tudo roda em um único processo. Contudo, quando dividimos a aplicação em vários serviços independentes, passamos a lidar com o desafio de saber onde cada serviço está executando, quantas instâncias ele tem, como rotear requisições e como lidar com balanceamento de carga.

Aqui temos os seguintes componentes:

- Service Discovery
- Roteamento (API Gateway) e Comunicação
- Balanceamento de Cargas

Vamos explorar melhor cada um desses componentes.

Service Discovery e Eureka

O **Service Discovery** (Descoberta de Serviço) é o mecanismo que **possibilita que os microsserviços "encontrem" uns aos outros** sem a necessidade de endereços fixos ou configurações manuais. Quando um serviço inicia, ele envia uma notificação ao sistema de registro, indicando algo como: "Estou vivo e posso ser acessado em hostX:portaY". Se esse serviço for replicado, cada réplica faz o mesmo procedimento, informando dados de localização distintos.

Outro serviço, ao precisar falar com o serviço disponibilizado, pergunta ao registro: "Onde encontro instâncias ativas do SubirService?". O registro devolve a lista de instâncias disponíveis, podendo até fornecer algum mecanismo de ordenação ou balanceamento.

Normalmente, os serviços enviam batidas de coração ("heartbeats") ou o próprio sistema de registro faz checagens de saúde, para confirmar que as instâncias ainda estão ativas. Se um serviço não responder no tempo esperado, ele é removido do registro (ou marcado como indisponível). Esse modelo garante que, caso uma instância fique indisponível, as próximas consultas já não apontem para ela, reduzindo falhas e timeouts.

Nesse contexto, temos o **Eureka**. Ele é um dos principais componentes de Service Discovery no ecossistema do Spring. Originalmente desenvolvido pela Netflix, o Eureka foi integrado ao Spring Cloud como parte do conjunto de ferramentas projetadas para lidar com arquiteturas de microsserviços. O Eureka resolve o problema de "onde está rodando o serviço X?" mediante um sistema de registro e consulta.

Para habilitarmos o Eureka, usamos a dependência:

```
<dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-starter-eureka-server</artifactId>
```

```
</dependency
```

Além da dependência, Aadiciona-se a anotação `@EnableEurekaServer` em uma classe de configuração (geralmente a classe principal). Nas propriedades (`application.properties`), define-se a porta em que o Eureka Server rodará (ex.: `server.port=8761`).

O Eureka conta com o **Eureka Server**, que atua como um "cadastro central" de todos os serviços, operando, por padrão, na porta 8761. A partir de uma API REST, ele disponibiliza funcionalidades de registro e remoção de instâncias, além de fornecer a lista das instâncias disponíveis para cada serviço – chamadas de *application name*.

Já os microsserviços executam um componente chamado de **Eureka Client**. Ele, ao iniciar, contacta o Eureka Server informando que está rodando, o endereço e a porta. Além disso, periodicamente envia heartbeats para indicar que continua vivo, e consulta o Eureka Server quando precisa encontrar outra aplicação.

Assim que um serviço com o Eureka Client sobe (por exemplo, "pedido-service"), ele detecta no application.properties (ou application.yml) qual o endereço do Eureka Server, e efetua uma chamada REST para registrar algo como:

JSON

```json
{
  "app": "pedido-service",
  "instanceId": "pedido-service:host-local:8080",
  "hostName": "host-local",
  "port": 8080,
  ...
}
```

Por padrão, o Eureka Client mantém uma cópia em cache das informações de registro, de modo que as requisições ao Eureka Server não sejam excessivas. Esse cache é atualizado periodicamente. Se, por algum motivo, o Eureka Server ficar indisponível por um tempo, as aplicações ainda conseguem utilizar a última versão conhecida das instâncias registradas.

O Eureka conta com o uso de Heartbeats, que podem ser configurados via `eureka.instance.lease-expiration-duration-in-seconds`. Além disso, o Eureka possui um mecanismo de "auto-preservação", que impede que a falta de atualizações de heartbeats em massa (por exemplo, devido a um problema de rede) remova todas as instâncias de uma só vez.

(FCC/TRT 22/2022) À classe principal da aplicação Spring Boot, um Técnico adicionou a anotação @EnableEurekaServer para fazer com que a aplicação atue como um servidor Eureka (Discovery Server). Em seguida, adicionou ao arquivo de configuração Maven pom.xml uma dependência, como mostrado abaixo.

```
<dependency>
..I..
</dependency>
```

Para que a dependência adicionada seja do servidor Spring Cloud Eureka, a lacuna I deve ser corretamente preenchida por

(A) <packageName>org.springframework.cloud</packageName> <resourceName>
spring-cloud-eureka-server</resourceName>
(B) <packageName>org.springframework.cloud</packageName>
<resourceName>spring-cloud-dependencies</resourceName>
(C) <groupId>spring.cloud.framework</groupId>
<artifactId>eureka-discovery-server</artifactId>
(D) compile('org.springframework.cloud:spring-cloud-starter-eureka-server')
(E) <groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-eureka-server</artifactId>

Comentários:

Como vimos, a dependência do Spring Cloud Eureka é feita da seguinte forma:

<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-eureka-server</artifactId>

Portanto, correta a letra E. *(Gabarito: Letra E)*

(FCC/TRT 15/2023) O Spring Eureka é uma implementação do padrão de registro de serviços que permite que os microsserviços se registrem automaticamente e se descubram uns aos outros. À porta padrão em que o servidor Eureka irá executar na aplicação Spring Boot e o arquivo em que essa porta é configurada são, respectivamente,

a) 80 e eureka-server. xml
b) 15000 e application-server.yml
c) 8080 e pom.xml
d) 8005 e eureca-registry.xml
e) 8761 e application.properties

Comentários:

A porta padrão do Eureka Server é a 8761, e suas configurações são inseridas no application.properties ou no application.yml. *(Gabarito: Letra E)*

Roteamento e Comunicação

O **roteamento e a comunicação** em sistemas distribuídos tratam de **como as requisições fluem entre microsserviços e como os serviços interagem uns com os outros**, sejam essas requisições iniciadas por um cliente externo ou originadas dentro do sistema. No contexto do Spring Cloud, essas funcionalidades são habilitadas por uma combinação de componentes e padrões que simplificam o gerenciamento de rotas, balanceamento de carga, autenticação e manipulação de dados em trânsito.

API Gateway

Aqui o componente mais importante é a **API Gateway**, que é a responsável pelo **roteamento** das comunicações. O roteamento define o caminho que as requisições seguem, desde a origem até o serviço correto. Isso inclui decisões sobre qual serviço receberá a requisição, como as rotas são resolvidas e como as respostas retornam ao cliente.

A API Gateway atua como o "porteiro" da aplicação, recebendo todas as requisições externas e roteando-as para o serviço apropriado. Em uma arquitetura de microsserviços, cada serviço atende a um domínio específico (pedidos, usuários, pagamentos, estoque etc.) e muitas vezes executa em múltiplas instâncias. Se cada requisição externa precisasse conhecer o endereço de cada serviço ou instância, o grau de complexidade e a necessidade de atualizações constantes seriam enormes.

O API Gateway resolve esse problema ao fornecer um ponto de entrada único, por meio do qual:

- O cliente externo envia a requisição apenas para o Gateway, sem precisar conhecer todos os detalhes da topologia interna.
- O Gateway determina qual serviço deve receber a requisição, possivelmente consultando um Service Discovery (como Eureka).
- O serviço interno processa a requisição e retorna a resposta, a qual o Gateway encaminha de volta ao cliente.

Cada chamada HTTP pode ser mapeada para um destino específico (ex.: /produtos/** vai para o serviço de produtos), simplificando a organização das rotas. Os clientes não precisam conhecer portas, nomes de serviço ou lógica de descoberta interna. O Gateway esconde esses detalhes e efetua a resolução dinamicamente.

O **Spring Cloud Gateway** é a implementação nativa do ecossistema Spring para o padrão de API Gateway. Ele substituiu ou complementa ferramentas mais antigas, como o Zuul (inicialmente da Netflix). Seu funcionamento é estruturado em torno de **rotas** e **filtros**:

- **Rotas:** cada rota define um conjunto de predicados que identificam quais requisições serão capturadas (por exemplo, verificar se o caminho começa com /produtos, se o método HTTP é GET, se existe determinado cabeçalho etc.). Uma URI de destino, que pode ser um serviço registrado no Eureka (lb://produto-service) ou um endereço fixo (por exemplo, *http://meuservico.local*).
- **Filtros:** podem ser aplicados em várias fases do fluxo (antes de encaminhar, durante o encaminhamento, ou depois de receber a resposta). Alguns usos comuns incluem:
    - Autenticação e Autorização: validação de tokens JWT, checagem de permissões.
    - Modificações na Requisição: adicionar cabeçalhos, reescrever partes do caminho, inserir informações de tracking, entre outros.
    - Manipulação de Respostas: por exemplo, remover ou inserir cabeçalhos antes de devolver ao cliente.

Com a propriedade `uri:  lb://nomeDoServico`, o Spring Cloud Gateway faz uso do Spring Cloud LoadBalancer (ou outro mecanismo) para resolver o destino dinamicamente, consultando o Eureka (ou outro Service Discovery). Assim, se o “produto-service” tiver várias instâncias, a requisição poderá ser encaminhada a qualquer uma delas, balanceando a carga.

Zuul

Uma alternativa para o Cloud Gateway é o **Zuul**. Ele é uma biblioteca desenvolvida pela Netflix, amplamente utilizada como API Gateway em arquiteturas de microsserviços. Ele atua como um ponto de entrada único para todas as requisições externas, centralizando o roteamento, aplicando filtros e fornecendo funcionalidades essenciais para sistemas distribuídos. Embora o Zuul tenha sido descontinuado em favor de ferramentas mais modernas, como o Spring Cloud Gateway, ele ainda é relevante em muitos projetos existentes e em sistemas legados.

O Zuul funciona como um proxy reverso que recebe requisições de clientes externos e as encaminha para os microsserviços apropriados. Ele é construído em torno do conceito de **filtros**, que são pequenas unidades de lógica programável usadas para modificar ou monitorar as requisições e respostas.

Podemos criar filtros customizados, que podem ser aplicados em diferentes fases do ciclo da requisição:

- Pré-Filtros: executados antes de a requisição ser encaminhada ao serviço. Úteis para autenticação, log de requisições, ou modificações nos cabeçalhos.
- Pós-Filtros: executados após a resposta do serviço ser recebida, mas antes de ser enviada ao cliente.
- Filtros de Roteamento: usados para controlar como a requisição será roteada para o backend.
- Filtros de Erro: executados quando ocorre algum problema no processamento da requisição ou resposta.

A dependência para o Zuul pode ser definida no pom.xml da seguinte forma:

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-zuul</artifactId>
</dependency>
```

A anotação responsável pelo Zuul é a `@EnableZuulProxy`.

(FCC/TRT 23/2022) Para fazer um aplicativo criado com Spring Boot, em condições ideais, funcionar como um servidor Zuul Proxy deve-se anotar a classe principal com

a) @EnableZuulProxy
b) @ZuulServerApplication
c) @EnabledZuulEdgeServer
d) @ZuulEdgeServerOn
e) @ZuulProxyApplication

Comentários:

Como vimos, a anotação usada é o @EnableZuulProxy. *(Gabarito: Letra A)*

Balanceamento de Cargas

O **balanceamento de carga** é uma técnica que permite **distribuir requisições entre múltiplas instâncias de um mesmo serviço**, garantindo melhor utilização dos recursos, escalabilidade e maior disponibilidade. Em

uma arquitetura de microsserviços, é comum ter vários serviços capazes de responder ao mesmo tipo de requisição. O balanceador de carga entra em ação para decidir a qual instância específica cada requisição deve ir.

No Spring Cloud, o componente que lida com essa necessidade de forma nativa é o **Spring Cloud LoadBalancer**, que substitui soluções anteriores como o Ribbon (da Netflix). Ele é totalmente integrado ao ecossistema Spring Boot e à camada de Service Discovery, possibilitando balancear chamadas a serviços que estejam registrados em mecanismos como o Eureka ou Consul, sem exigir configuração manual de endereços IP ou portas.

Em linhas gerais, o balanceamento de carga pode ocorrer em dois principais modos:

- **Client-side:** o próprio cliente (por exemplo, o microsserviço chamador) recebe do Service Discovery a lista de instâncias disponíveis para o serviço de destino. Ele escolhe qual instância chamar, usando alguma estratégia (round-robin, peso, menor latência etc.).
- Server-side: a requisição chega a um componente externo ao cliente (um proxy, por exemplo), que decide para qual instância encaminhar. O Spring Cloud Gateway, ou um balanceador da nuvem (ELB na AWS, por exemplo), pode cumprir esse papel, decidindo o destino conforme regras de roteamento e disponibilidade.

O **Spring Cloud LoadBalancer segue o modo client-side**. Por padrão, o round-robin é utilizado, mas o Spring Cloud LoadBalancer permite configurar classes personalizadas que implementam a interface `ReactorServiceInstanceLoadBalancer`, por exemplo, possibilitando:

- Random: escolha aleatória de instância.
- Weighted: onde algumas instâncias têm “peso” maior.
- Lowest Response Time: escolher a instância que responde mais rápido (geralmente requer algum monitoramento de latência).

Resiliência, Observabilidade e Mensageria

Em sistemas distribuídos, cada microsserviço roda em seu próprio processo, podendo existir inúmeras instâncias de um mesmo serviço e uma quantidade ainda maior de serviços diferentes que cooperam para entregar as funcionalidades da aplicação. Esse cenário oferece grandes vantagens — especialmente escalabilidade e independência de desenvolvimento — mas traz novos problemas, como maior suscetibilidade a falhas e dificuldades de rastreamento de requisições.

Para lidar com esses desafios, o Spring Cloud conta com recursos voltados para resiliência, observabilidade (que envolve, dentre outros, o monitoramento, o rastreamento e o logging) e mensageria (comunicação assíncrona). Vamos ver os principais recursos oferecidos.

Circuit Breakers

Em uma arquitetura de microsserviços, é inevitável que um serviço A chame um serviço B em algum momento. Entretanto, o serviço B pode ter problemas de performance ou estar totalmente fora do ar. Se o A continuar tentando se comunicar de forma insistente com um serviço que não responde ou responde lentamente, isso pode ocasionar *cascading failures* — ou seja, efeitos em cadeia que podem derrubar ou degradar todo o sistema.

Para evitar essa situação, o padrão **Circuit Breaker** age como um disjuntor elétrico: se o serviço A notar que o serviço B não está respondendo de maneira consistente, ele "abre o disjuntor" e passa a não encaminhar mais requisições para o B por um período (retornando um erro imediato ou um *fallback*), até que exista indício de que o B se recuperou. Essa mecânica poupa recursos e evita que múltiplos serviços entrem em colapso devido à dependência de um serviço indisponível.

O circuito começa "fechado", indicando que as chamadas podem fluir livremente para o serviço B. As requisições são encaminhadas e, se derem certo, tudo permanece fechado. Se um número ou percentual limite de falhas consecutivas (timeout, exceção, etc.) for atingido dentro de um intervalo, o Circuit Breaker "abre" o circuito.

Enquanto o circuito estiver aberto, as chamadas não são encaminhadas para o serviço B, retornando imediatamente uma resposta de fallback ou erro customizado. Isso impede que o serviço A fique preso em timeouts recorrentes, liberando recursos para outras ações.

Após expirar um tempo de espera (ou "timeout" do disjuntor), o Circuit Breaker passa para um estado "meio aberto". Nesse estado, algumas requisições são encaminhadas ao serviço B para testar se ele voltou a ficar estável. Se as tentativas tiverem sucesso, o disjuntor é fechado novamente. Se falharem, o circuito volta a ficar aberto.

O **Spring Cloud Circuit Breaker** oferece um conjunto de abstrações para implementar esse padrão de forma uniforme, independentemente da biblioteca subjacente (Resilience4j, Hystrix, Sentinel, etc.). Atualmente, Resilience4j é a escolha moderna e recomendada para a maioria dos casos. Ele fornece anotações ou APIs para configurar o comportamento do Circuit Breaker (limite de falhas, tempo de espera, fallback, etc.).

Spring Cloud Sleuth

O **Spring Cloud Sleuth** é uma biblioteca do ecossistema Spring Cloud projetada para lidar com o **rastreamento distribuído de requisições** em sistemas distribuídos, como arquiteturas baseadas em microsserviços. Ele fornece suporte para monitorar o caminho que uma requisição percorre enquanto atravessa diferentes serviços, facilitando a identificação de gargalos, erros e dependências em tempo de execução.

Em sistemas monolíticos, é fácil rastrear uma requisição porque tudo ocorre dentro de um único processo. Em microsserviços, no entanto, uma única transação pode envolver vários serviços diferentes, cada um rodando em processos ou máquinas distintas. O Sleuth resolve esse problema ao **adicionar metadados únicos a cada requisição** e propagá-los entre os serviços.

Para adicionar a dependência ao pom.xml, usamos:

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-sleuth</artifactId>
</dependency>
```

O Sleuth trabalha com dois conceitos principais:

- **Trace:** representa uma transação completa. Um trace ID identifica o caminho de uma requisição por todos os serviços envolvidos.
- **Span:** representa uma unidade de trabalho dentro de um trace. Cada serviço (ou operação) tem seu próprio span ID, que está associado ao trace ID.

Um trace pode ser visto como um conjunto de spans que se conectam entre si para formar o fluxo completo de uma requisição. Usualmente associamos esses pacotes ao Zipkin, uma ferramenta responsável por criar os painéis de visualização, coletando e armazenando os dados de rastreamento.

Spring Cloud Stream

O Spring Cloud Stream é um módulo do Spring Cloud projetado para simplificar a criação de aplicações baseadas em **event-driven architecture (EDA)**, ou arquitetura orientada a eventos. Ele fornece uma abstração para integrar microsserviços com sistemas de mensageria, como Apache Kafka e RabbitMQ, facilitando o envio, processamento e consumo de mensagens entre serviços.

Em uma arquitetura orientada a eventos, serviços se comunicam de maneira assíncrona, publicando e consumindo eventos em tópicos ou filas. Isso reduz o acoplamento entre os serviços e aumenta a resiliência do sistema, já que os serviços podem continuar operando independentemente se o produtor ou consumidor de mensagens estiver temporariamente indisponível.

O Stream trabalha com ligações, ou ***bindings*** e ***binders***. O Spring Cloud Stream conecta os métodos da aplicação com destinos de mensagens (como tópicos no Kafka ou filas no RabbitMQ). Os *bindings* atuam como "pontes" entre o código do desenvolvedor e os sistemas de mensageria. Os binders são implementações que permitem ao Spring Cloud Stream se integrar com diferentes sistemas de mensageria.

A dependência para o Stream varia de acordo com o tipo de ferramenta de mensageria utilizada. Para o Kafka, por exemplo, usamos:

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-stream-kafka</artifactId>
</dependency>
```

Spring Cloud Bus

O **Spring Cloud Bus** é uma ferramenta projetada para **conectar diferentes instâncias de microsserviços usando um sistema de mensageria**, como RabbitMQ ou Apache Kafka. Seu objetivo principal é propagar eventos ou atualizações de configuração de forma eficiente entre múltiplos serviços distribuídos, garantindo que todos permaneçam sincronizados.

Em sistemas distribuídos, especialmente aqueles com muitas instâncias de serviços, gerenciar alterações e eventos pode ser desafiador. O Spring Cloud Bus automatiza essas tarefas ao permitir que mensagens sejam enviadas a todas as instâncias conectadas, funcionando como um barramento (bus) entre os serviços.

O Spring Cloud Bus e o Spring Cloud Stream são ferramentas complementares no ecossistema Spring Cloud. O Spring Cloud Bus é projetado para propagar eventos de sistema, como atualizações de configuração ou mensagens administrativas, entre instâncias de microsserviços usando um barramento comum (tipicamente RabbitMQ ou Kafka), com foco em sincronização e gerenciamento distribuído.

Já o Spring Cloud Stream fornece uma abstração para criar aplicações orientadas a eventos, facilitando a produção e o consumo de mensagens entre serviços de forma assíncrona, promovendo integração desacoplada em sistemas distribuídos.

Enquanto o Bus é mais usado para comunicação entre instâncias do próprio sistema, o Stream é voltado para integrar serviços ou sistemas externos em fluxos baseados em mensageria.

Em conjunto com o Spring Cloud Config, o Bus permite atualizar dinamicamente as configurações de todas as instâncias de um serviço quando o Config Server detecta mudanças nos arquivos de configuração.

O Bus também depende da ferramenta de implementação para definirmos sua dependência. Para o RabbitMQ, temos:

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-bus-amqp</artifactId>
</dependency>
```

QUESTÕES COMENTADAS

Spring Cloud

01. (FCC/TRT 4/2022) Para gerenciar a configuração de um panorama do sistema de microsserviços, o Spring Cloud contém o Spring Cloud Config, que fornece o gerenciamento centralizado de arquivos de configuração. O Spring Cloud Config

a) oferece suporte à criptografia de informações confidenciais na configuração, exceto de credenciais.
b) suporta o armazenamento de arquivos de configuração em repositório Git, por exemplo, no GitHub.
c) não permite separar as partes comuns da configuração das partes específicas de microsserviços.
d) não suporta o armazenamento de arquivos de configuração em uma base de dados banco de dados JDBC.
e) não suporta o armazenamento de arquivos de configuração em um sistema de arquivos local (local filesystem).

Comentários:

Vamos analisar a alternativas.

a) Errado. O Spring Cloud Config oferece suporte à criptografia de informações confidenciais, incluindo credenciais. Ele permite criptografar propriedades sensíveis nos arquivos de configuração usando chaves assimétricas, garantindo a segurança dos dados.
b) Certo. O Spring Cloud Config suporta o armazenamento de arquivos de configuração em repositórios Git, incluindo serviços como GitHub, GitLab e Bitbucket, além de repositórios privados.
c) Errado. O Spring Cloud Config permite separar as partes comuns da configuração (compartilhadas por vários microsserviços) das partes específicas de cada serviço. Isso pode ser feito organizando os arquivos por perfis, como application.yml (configurações comuns) e service-name.yml (configurações específicas).
d) Errado. O Spring Cloud Config suporta o armazenamento de arquivos de configuração em bases de dados JDBC, permitindo que configurações sejam lidas diretamente de tabelas no banco de dados.
e) Errado. O Spring Cloud Config suporta o armazenamento de arquivos de configuração em sistemas de arquivos locais. Isso pode ser útil em cenários de desenvolvimento ou para soluções simples que não dependem de repositórios remotos.

Portanto, correta a letra B.

Gabarito: Letra B

Spring Cloud Eureka

02. (FCC/TRT 21/2023) O Spring Cloud Netflix Eureka, em uma arquitetura de microsserviços,

a) fornece uma maneira padronizada de descrever as funcionalidades de uma API da web ou de um microsserviço.
b) permite que os serviços se registrem em um servidor de descoberta e descubram uns aos outros usando nomes lógicos em vez de endereços IP fixos.
c) é responsável pela execução de operações de negócios nos microsserviços.
d) permite que as solicitações de clientes sejam encaminhadas para os microsserviços apropriados com base em critérios, como roteamento, filtragem, autenticação e autorização.
e) permite armazenar e recuperar as configurações dos serviços em um repositório central, como Git ou Subversion.

Comentários:

O Spring Cloud Eureka é um sistema de descoberta de serviços que busca criar uma abstração em relação aos endereços lógicos dos serviços, num ecossistema de microsserviços. Ele permite que serviços se registrem a partir de nomes, ao invés de IPs, facilitando o processo de descoberta. Nesse sentido, a alternativa que mais se alinha à explicação do componente é a letra B.

Gabarito: Letra B

---

03. (FCC/MPE PB/2023) No Spring Cloud, a anotação compatível com o servidor de registro de serviço Eureka usada para habilitar a descoberta de serviços em um ambiente distribuído é a anotação

a) @EnableDiscoveryClient.
b) @FeignDiscoveryClient.
c) @EnableConfigServer.
d) @EnableCircuitBreaker.
e) @EnableEurekaServiceDiscovery.

Comentários:

Queremos a anotação responsável por habilitar a descoberta de serviços no Eureka. Vamos analisar as alternativas:

a) @EnableDiscoveryClient: Habilita a funcionalidade de descoberta de serviços (Service Discovery) no Spring Cloud, conectando o serviço a um registro, como Eureka ou Consul.
b) @FeignDiscoveryClient: Não existe. Para Feign e Service Discovery, usa-se geralmente @EnableFeignClients combinado com @EnableDiscoveryClient.
c) @EnableConfigServer: Ativa o Spring Cloud Config Server, permitindo que ele sirva configurações centralizadas para múltiplos serviços.
d) @EnableCircuitBreaker: Habilita o suporte a Circuit Breakers no Spring Cloud, permitindo resiliência em chamadas a serviços externos.
e) @EnableEurekaServiceDiscovery: Não existe. O equivalente é @EnableEurekaClient ou @EnableDiscoveryClient para habilitar o Eureka como mecanismo de descoberta.

Portanto, correta a letra A.

Gabarito: Letra A

---

04. (FCC/TRT 22/2022) À classe principal da aplicação Spring Boot, um Técnico adicionou a anotação @EnableEurekaServer para fazer com que a aplicação atue como um servidor Eureka (Discovery Server). Em seguida, adicionou ao arquivo de configuração Maven pom.xml uma dependência, como mostrado abaixo.

```
<dependency>
  ..I..
</dependency>
```

Para que a dependência adicionada seja do servidor Spring Cloud Eureka, a lacuna I deve ser corretamente preenchida por

a) <packageName>org.springframework.cloud</packageName> <resourceName> spring-cloud-eureka-server</resourceName>
b) <packageName>org.springframework.cloud</packageName> <resourceName>spring-cloud-dependencies</resourceName>
c) <groupId>spring.cloud.framework</groupId> <artifactId>eureka-discovery-server</artifactId>
d) compile('org.springframework.cloud:spring-cloud-starter-eureka-server')
e) <groupId>org.springframework.cloud</groupId> <artifactId>spring-cloud-starter-eureka-server</artifactId>

Comentários:

A dependência para habilitarmos o Spring Cloud Eureka é:

```
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-eureka-server</artifactId>
```

Portanto, correta a letra E.

Gabarito: Letra E

---

05. (FCC/TRT 21/2023) O Spring Cloud Netflix Eureka, em uma arquitetura de microsserviços,

a) fornece uma maneira padronizada de descrever as funcionalidades de uma API da web ou de um microsserviço.
b) permite que os serviços se registrem em um servidor de descoberta e descubram uns aos outros usando nomes lógicos em vez de endereços IP fixos.
c) é responsável pela execução de operações de negócios nos microsserviços.
d) permite que as solicitações de clientes sejam encaminhadas para os microsserviços apropriados com base em critérios, como roteamento, filtragem, autenticação e autorização.
e) permite armazenar e recuperar as configurações dos serviços em um repositório central, como Git ou Subversion.

Comentários:

Vamos às alternativas.

a) Errado. Descrever funcionalidades de uma API da web ou de um microsserviço é papel de ferramentas como Swagger ou OpenAPI, não do Spring Cloud Netflix Eureka.
b) Certo. O Spring Cloud Netflix Eureka é um serviço de descoberta (Service Discovery). Ele permite que os serviços se registrem em um servidor central (Eureka Server) e descubram outros serviços usando nomes lógicos, eliminando a necessidade de endereços IP fixos e facilitando a escalabilidade e a flexibilidade da infraestrutura.
c) Errado. A execução de operações de negócios é responsabilidade dos próprios microsserviços, que implementam as regras e lógicas de negócio. O Eureka não desempenha essa função.
d) Errado. Encaminhar requisições com base em critérios como roteamento, autenticação e autorização é o papel de um API Gateway, como o Netflix Zuul ou o Spring Cloud Gateway, não do Eureka.
e) Errado. Armazenar e recuperar configurações centralizadas é a função do Spring Cloud Config, que pode usar repositórios como Git ou Subversion para gerenciar os arquivos de configuração.

Portanto, correta a letra B.

Gabarito: Letra B

---

06. (FCC/TJA BA/2023) Em um ambiente de desenvolvimento em condições ideais, uma Analista possui um aplicativo Spring Boot chamado tjuba que deseja registrar no Eureka Server. Para indicar que o aplicativo deve usar o Eureka Server em http://localhost:8761/eureka/ como seu servidor padrão de descoberta de serviços, ela deve Incluir no arquivo application.properties a instrução

a) eureka.i nstance.defaultZone.locale=http://localhost:8761/eureka/
b) eureka.instance.instanceUrl.localeDefaultZone=http://localhost:8761/eureka/
c) eureka.client.fetchRegistr y.defaultZoneUrl=http://localhost:8761/eureka/
d) eureka.client.register-with-eureka.localeZone=http://localhost:8761/eureka/
e) eureka.client.serviceUrl.defaultZone=http://localhost:8161/eureka/

Comentários:

Para habilitarmos o Eureka como servidor padrão, usamos a seguinte instrução no application.proprerties: eureka.client.serviceUrl.defaultZone=http://localhost:8161/eureka/. Portanto, correta a letra E.

Gabarito: Letra E

---

07. (FCC/TRT 18/2023) Uma Analista criou um projeto Maven e incluiu a dependência abaixo no arquivo Project Object Model (POM).

```
<dependency>
        <groupId>org.springframework.cloud</groupId>
        ...I...
</dependency>
```

Como deseja implementar um servidor Eureka para registro de serviço, em condições ideais, a lacuna I deve ser corretamente preenchida por:

a) <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
b) <elementId>spring-cloud-starter-eureka-discobery-server</elementId>
c) <artifact>spring-cloud-starter-parent-eureka-server</artifact>
d) <elementId>spring-cloud-starter-eureka-server</elementId>
e) <artifact>spring-cloud-service-discovery-starter</artifact>

Comentários:

A questão cobrou uma notação “alternativa” para implementarmos o Eureka, um pouco mais antiga – apesar de ser uma questão relativamente recente. Ela usa a seguinte anotação para o artefato:

spring-cloud-starter-netflix-eureka-server

Essa dependência é a versão mais antiga e diretamente vinculada ao código original do Netflix Eureka.

Gabarito: Letra A

---

08. (FCC/TRT 23/2022) Considere o código fonte a seguir.

```
public class Application {
    public static void main(String[] args) {
    new SpringApplicationBuilder(Application.class).web(true).run(args);
    }
}
```

Em condições ideais, para este código ser de um servidor Eureka mínimo, antes da declaração da classe devem ser colocadas as anotações

a) @EnableNetflixEureka e @EnableDiscoveryClient
b) @EnableEurekaConfiguration e @EnableRegistryServer
c) @SpringBootServer e @SpringBootEureka
d) @SpringBootApplication e @EnableEurekaServer
e) @EurekaDiscoveryClient e @EurekaServiceRegistry

Comentários:

O código apresentado utiliza o SpringApplicationBuilder para inicializar uma aplicação Spring Boot com um contexto web habilitado. Para que esse código represente um servidor Eureka mínimo, é necessário incluir a anotação correta que habilite o comportamento de um Eureka Server.

Para isso, primeiro precisamos marcar a classe como uma aplicação do Spring, com @SpringBootApplication. Em seguida, habilitamos as funcionalidades do Eureka Server com @EnableEurekaServer, permitindo registrar e descobrir os serviços.

Gabarito: Letra D

Zuul

09. (FCC/TRT 4/2022) Em um cenário de sistema de microsserviços um técnico deseja expor alguns dos microsserviços para acesso externo e ocultar outros. Como ele está usando o Spring Cloud, para proteger os microsserviços expostos contra solicitações de clientes malintencionados, ele pode usar o

a) Galaxy Eureka como edge server.
b) Apache Toncat como service discovery.
c) Apache Wildfly como security server.
d) Netflix Zuul como edge server.
e) Swagger como security server.

Comentários:

Vamos analisar as alternativas.

a) Errado. Galaxy Eureka não existe. O correto seria Netflix Eureka, que é usado como serviço de descoberta (Service Discovery), e não como edge server ou mecanismo de proteção de microsserviços.
b) Errado. O Apache Tomcat é um servidor de aplicação que pode hospedar serviços web, mas não oferece funcionalidades de descoberta de serviços (service discovery). Ele também não é usado como mecanismo de segurança em sistemas distribuídos.
c) Errado. O Apache WildFly é um servidor de aplicação, mas não é um "security server" no contexto de Spring Cloud ou sistemas de microsserviços. Ele não fornece funcionalidades específicas para proteger microsserviços.
d) Certo. O Netflix Zuul pode atuar como um edge server em arquiteturas de microsserviços. Ele funciona como um API Gateway, permitindo expor alguns serviços externamente, ocultar outros e implementar filtros de segurança para proteger os microsserviços contra acessos não autorizados ou maliciosos.
e) Errado. O Swagger é uma ferramenta para documentação e teste de APIs REST, mas não é um servidor de segurança. Ele não oferece funcionalidades de proteção para microsserviços.

Portanto, correta a letra D.

Gabarito: Letra D

---

10. (FAURGS/HCPA/2020) Dentro de uma solução estruturada numa arquitetura de microsserviços e implementada utilizando Spring Cloud, qual o principal papel do Spring Cloud Netflix Zuul?

a) Fornecer a configuração dos demais componentes da arquitetura.
b) Permitir que o front-end chame os serviços do back-end.
c) Depurar a comunicação entre microsserviços.
d) Permitir comunicação assíncrona entre os microsserviços.
e) Permitir que o back-end execute consultas na camada de persistência.

Comentários:

O Zuul atua como uma API Gateway, servindo como ponto de entrada para as requisições. Com isso em mente, vamos analisar as alternativas.

a) Errado. Fornecer a configuração dos demais componentes é o papel do Spring Cloud Config, que centraliza e distribui configurações para os serviços. O Netflix Zuul não desempenha essa função.
b) Certo. O principal papel do Spring Cloud Netflix Zuul é atuar como um API Gateway ou edge server, permitindo que o front-end (ou outros clientes externos) acessem os serviços do back-end. Ele também roteia requisições e pode aplicar políticas de segurança, autenticação e autorização.
c) Errado. Depurar a comunicação entre microsserviços é uma tarefa geralmente facilitada por ferramentas de rastreamento distribuído, como o Spring Cloud Sleuth e o Zipkin, não pelo Zuul.
d) Errado. A comunicação assíncrona entre microsserviços é tratada por ferramentas como o Spring Cloud Stream, que integra sistemas de mensageria como RabbitMQ ou Kafka. O Zuul não lida diretamente com esse tipo de comunicação.
e) Errado. Permitir que o back-end execute consultas na camada de persistência não é responsabilidade do Zuul. Isso é tratado por frameworks como Spring Data e JPA, que gerenciam o acesso ao banco de dados.

Sendo assim, correta a letra B.

Gabarito: Letra B

---

11. (FCC/TRT 23/2022) Para fazer um aplicativo criado com Spring Boot, em condições ideais, funcionar como um servidor Zuul Proxy deve-se anotar a classe principal com

a) @EnableZuulProxy
b) @ZuulServerApplication
c) @EnabledZuulEdgeServer
d) @ZuulEdgeServerOn
e) @ZuulProxyApplication

Comentários:

Para marcamos uma classe como servidor Zuul, usamos a anotação @EnableZuulProxy.

Gabarito: Letra A

---

12. (CEBRASPE/STJ/2024) Em relação à linguagem de programação Java, à arquitetura distribuída de microsserviços e à biblioteca Flyway, julgue os próximos itens

Zuul é uma solução que permite a realização de roteamento dinâmico e monitoramento, e seus filtros são capazes de atuar na segurança por meio da identificação de requisitos de autenticação para cada recurso e da rejeição de solicitações que não os satisfaçam.

Comentários:

O item está correto. O Zuul, como parte do ecossistema Netflix e integrado ao Spring Cloud, é uma solução projetada para atuar como um API Gateway. Ele oferece roteamento dinâmico, monitoramento e a capacidade de implementar filtros customizados. Esses filtros podem ser usados para diversas finalidades, incluindo segurança, como verificar requisitos de autenticação e autorização para cada recurso. Se uma solicitação não atender aos critérios definidos nos filtros, o Zuul pode rejeitá-la antes mesmo que ela alcance o serviço de destino.

Gabarito: Certo

LISTA DE QUESTÕES

Spring Cloud

01. (FCC/TRT 4/2022) Para gerenciar a configuração de um panorama do sistema de microsserviços, o Spring Cloud contém o Spring Cloud Config, que fornece o gerenciamento centralizado de arquivos de configuração. O Spring Cloud Config

a) oferece suporte à criptografia de informações confidenciais na configuração, exceto de credenciais.
b) suporta o armazenamento de arquivos de configuração em repositório Git, por exemplo, no GitHub.
c) não permite separar as partes comuns da configuração das partes específicas de microsserviços.
d) não suporta o armazenamento de arquivos de configuração em uma base de dados banco de dados JDBC.
e) não suporta o armazenamento de arquivos de configuração em um sistema de arquivos local (local filesystem).

Spring Cloud Eureka

02. (FCC/TRT 21/2023) O Spring Cloud Netflix Eureka, em uma arquitetura de microsserviços,

a) fornece uma maneira padronizada de descrever as funcionalidades de uma API da web ou de um microsserviço.
b) permite que os serviços se registrem em um servidor de descoberta e descubram uns aos outros usando nomes lógicos em vez de endereços IP fixos.
c) é responsável pela execução de operações de negócios nos microsserviços.
d) permite que as solicitações de clientes sejam encaminhadas para os microsserviços apropriados com base em critérios, como roteamento, filtragem, autenticação e autorização.
e) permite armazenar e recuperar as configurações dos serviços em um repositório central, como Git ou Subversion.

03. (FCC/MPE PB/2023) No Spring Cloud, a anotação compatível com o servidor de registro de serviço Eureka usada para habilitar a descoberta de serviços em um ambiente distribuído é a anotação

a) @EnableDiscoveryClient.
b) @FeignDiscoveryClient.
c) @EnableConfigServer.
d) @EnableCircuitBreaker.
e) @EnableEurekaServiceDiscovery.

04. (FCC/TRT 22/2022) À classe principal da aplicação Spring Boot, um Técnico adicionou a anotação @EnableEurekaServer para fazer com que a aplicação atue como um servidor Eureka (Discovery Server). Em seguida, adicionou ao arquivo de configuração Maven pom.xml uma dependência, como mostrado abaixo.

```
<dependency>
  ..I..
</dependency>
```

Para que a dependência adicionada seja do servidor Spring Cloud Eureka, a lacuna I deve ser corretamente preenchida por

a) <packageName>org.springframework.cloud</packageName> <resourceName> spring-cloud-eureka-server</resourceName>
b) <packageName>org.springframework.cloud</packageName> <resourceName>spring-cloud-dependencies</resourceName>
c) <groupId>spring.cloud.framework</groupId> <artifactId>eureka-discovery-server</artifactId>
d) compile('org.springframework.cloud:spring-cloud-starter-eureka-server')
e) <groupId>org.springframework.cloud</groupId> <artifactId>spring-cloud-starter-eureka-server</artifactId>

05. (FCC/TRT 21/2023) O Spring Cloud Netflix Eureka, em uma arquitetura de microsserviços,

a) fornece uma maneira padronizada de descrever as funcionalidades de uma API da web ou de um microsserviço.
b) permite que os serviços se registrem em um servidor de descoberta e descubram uns aos outros usando nomes lógicos em vez de endereços IP fixos.
c) é responsável pela execução de operações de negócios nos microsserviços.
d) permite que as solicitações de clientes sejam encaminhadas para os microsserviços apropriados com base em critérios, como roteamento, filtragem, autenticação e autorização.
e) permite armazenar e recuperar as configurações dos serviços em um repositório central, como Git ou Subversion.

06. (FCC/TJA BA/2023) Em um ambiente de desenvolvimento em condições ideais, uma Analista possui um aplicativo Spring Boot chamado tjuba que deseja registrar no Eureka Server. Para indicar que o aplicativo deve usar o Eureka Server em http://localhost:8761/eureka/ como seu servidor padrão de descoberta de serviços, ela deve Incluir no arquivo application.properties a instrução

a) eureka.i nstance.defaultZone.locale=http://localhost:8761/eureka/
b) eureka.instance.instanceUrl.localeDefaultZone=http://localhost:8761/eureka/
c) eureka.client.fetchRegistr y.defaultZoneUrl=http://localhost:8761/eureka/
d) eureka.client.register-with-eureka.localeZone=http://localhost:8761/eureka/
e) eureka.client.serviceUrl.defaultZone=http://localhost:8161/eureka/

07. (FCC/TRT 18/2023) Uma Analista criou um projeto Maven e incluiu a dependência abaixo no arquivo Project Object Model (POM).

```
<dependency>
        <groupId>org.springframework.cloud</groupId>
```

```
        ...I...
    </dependency>
```

Como deseja implementar um servidor Eureka para registro de serviço, em condições ideais, a lacuna I deve ser corretamente preenchida por:

a) <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
b) <elementId>spring-cloud-starter-eureka-discobery-server</elementId>
c) <artifact>spring-cloud-starter-parent-eureka-server</artifact>
d) <elementId>spring-cloud-starter-eureka-server</elementId>
e) <artifact>spring-cloud-service-discovery-starter</artifact>

08. (FCC/TRT 23/2022) Considere o código fonte a seguir.

```
public class Application {
    public static void main(String[] args) {
    new SpringApplicationBuilder(Application.class).web(true).run(args);
    }
}
```

Em condições ideais, para este código ser de um servidor Eureka mínimo, antes da declaração da classe devem ser colocadas as anotações

a) @EnableNetflixEureka e @EnableDiscoveryClient
b) @EnableEurekaConfiguration e @EnableRegistryServer
c) @SpringBootServer e @SpringBootEureka
d) @SpringBootApplication e @EnableEurekaServer
e) @EurekaDiscoveryClient e @EurekaServiceRegistry

Zuul

09. (FCC/TRT 4/2022) Em um cenário de sistema de microsserviços um técnico deseja expor alguns dos microsserviços para acesso externo e ocultar outros. Como ele está usando o Spring Cloud, para proteger os microsserviços expostos contra solicitações de clientes malintencionados, ele pode usar o

a) Galaxy Eureka como edge server.
b) Apache Toncat como service discovery.
c) Apache Wildfly como security server.
d) Netflix Zuul como edge server.
e) Swagger como security server.

10. (FAURGS/HCPA/2020) Dentro de uma solução estruturada numa arquitetura de microsserviços e implementada utilizando Spring Cloud, qual o principal papel do Spring Cloud Netflix Zuul?

a) Fornecer a configuração dos demais componentes da arquitetura.
b) Permitir que o front-end chame os serviços do back-end.

c) Depurar a comunicação entre microsserviços.
d) Permitir comunicação assíncrona entre os microsserviços.
e) Permitir que o back-end execute consultas na camada de persistência.

11. (FCC/TRT 23/2022) Para fazer um aplicativo criado com Spring Boot, em condições ideais, funcionar como um servidor Zuul Proxy deve-se anotar a classe principal com

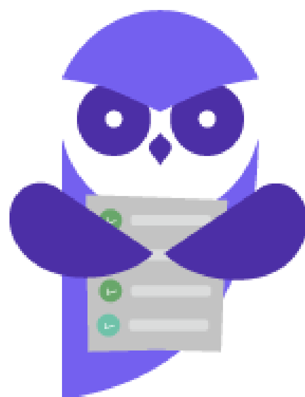
a) @EnableZuulProxy
b) @ZuulServerApplication
c) @EnabledZuulEdgeServer
d) @ZuulEdgeServerOn
e) @ZuulProxyApplication

12. (CEBRASPE/STJ/2024) Em relação à linguagem de programação Java, à arquitetura distribuída de microsserviços e à biblioteca Flyway, julgue os próximos itens

Zuul é uma solução que permite a realização de roteamento dinâmico e monitoramento, e seus filtros são capazes de atuar na segurança por meio da identificação de requisitos de autenticação para cada recurso e da rejeição de solicitações que não os satisfaçam.

GABARITO

GABARITO



1. Letra B
2. Letra B
3. Letra A
4. Letra E
5. Letra B
6. Letra E
7. Letra A
8. Letra D
9. Letra D
10. Letra B
11. Letra A
12. Ceto

MAP STRUCT

Conceitos Gerais

O **MapStruct** é uma biblioteca Java projetada para simplificar o **mapeamento entre objetos de** diferentes tipos, gerando automaticamente o código necessário para realizar essas conversões. Em muitos projetos, é comum precisarmos transferir dados de um objeto para outro, como converter uma entidade do banco de dados (Entity) para um objeto de transporte de dados (DTO) ou vice-versa. Fazer esse mapeamento manualmente, além de ser repetitivo, pode introduzir erros e dificultar a manutenção do código.

Por exemplo, imagine um cenário em que uma aplicação possui dezenas ou até centenas de classes de dados. O mapeamento manual entre esses objetos exigiria muitos métodos de conversão, o que aumentaria a quantidade de código boilerplate (código repetitivo). É nesse contexto que o MapStruct se destaca.

A principal vantagem do **MapStruct** é que ele **utiliza anotações** para definir as regras de mapeamento e, com base nisso, gera o código de mapeamento em tempo de compilação. Essa abordagem garante alta performance, já que evita o uso de reflexão, comum em outras bibliotecas, e permite um código mais eficiente e fácil de depurar.

Além disso, o MapStruct integra-se perfeitamente com projetos que utilizam frameworks como Spring, CDI ou Lombok, tornando-o uma escolha natural para quem já está no ecossistema Java moderno. Com uma curva de aprendizado baixa e um impacto significativo na redução do esforço manual, o MapStruct é uma ferramenta indispensável para desenvolvedores que buscam eficiência e simplicidade ao lidar com conversões de objetos.

O MapStruct gera automaticamente o código de mapeamento durante a compilação. Isso significa que o código gerado:

- É altamente otimizado, pois não depende de reflexão em tempo de execução.
- Pode ser inspecionado pelo desenvolvedor, facilitando a depuração e a compreensão de como os mapeamentos são realizados.
- Reduz o risco de erros em mapeamentos manuais, garantindo consistência.

As conversões realizadas podem ser **implícitas**, onde o MapStruct faz o mapeamento automático de campos com o mesmo nome e tipo entre dois objetos, ou a partir de **conversões explícitas**, nos casos em que temos nomes ou tipos diferentes, usando anotações como `@Mapping` para definir as regras de conversão.

Configuração e Dependências

O MapStruct é uma biblioteca fácil de configurar, especialmente quando usada com Maven ou Gradle. A configuração envolve adicionar as dependências corretas ao projeto e garantir que o processador de anotações esteja ativado para gerar o código de mapeamento automaticamente durante a compilação.

Vamos focar na implementação usando o Maven – é necessário listar duas dependências no `pom.xml`:

- **MapStruct API:** Contém as anotações e interfaces necessárias para definir os mapeamentos.
- **Processor:** Responsável por processar as anotações e gerar o código de mapeamento durante a compilação.

Ficamos com as seguintes dependências:

```
<dependencies>
    <!-- Dependência principal do MapStruct -->
    <dependency>
        <groupId>org.mapstruct</groupId>
        <artifactId>mapstruct</artifactId>
        <version>1.5.5.Final</version>
    </dependency>

    <!-- Processador de anotações do MapStruct -->
    <dependency>
        <groupId>org.mapstruct</groupId>
        <artifactId>mapstruct-processor</artifactId>
        <version>1.5.5.Final</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

O escopo `provided` no processador indica que ele é necessário apenas durante a compilação, não em tempo de execução.

O *processor*, ou **Processador de Anotações**, é responsável por analisar as interfaces anotadas com `@Mapper` e gerar as classes de implementação correspondentes. Essa configuração é geralmente automática em IDEs modernas, mas, em alguns casos, pode ser necessário configurá-lo manualmente.

O MapStruct gera automaticamente a implementação das interfaces marcadas com `@Mapper`. O código gerado pode ser encontrado no diretório `target/generated-sources/annotations` (Maven) ou `build/generated/sources/annotationProcessor` (Gradle).

Componentes

O MapStruct é construído em torno de anotações e conceitos que tornam o mapeamento entre objetos eficiente e flexível. Esses componentes são os pilares para usar a biblioteca de maneira eficaz. Vamos detalhar os principais componentes.

@Mapper

A anotação `@Mapper` marca uma interface ou classe abstrata como mapeador. Ela informa ao MapStruct que uma implementação concreta dessa interface deve ser gerada durante a compilação.

Java

```java
@Mapper
public interface UsuarioMapper {
    UsuarioDTO toUsuarioDTO(Usuario usuario);
}
```

Nesse exemplo, a implementação `UsuarioMapperImpl` será gerada automaticamente. O método `toUsuarioDTO` mapeará os campos do objeto Usuario para `UsuarioDTO`.

@Mapping

A anotação `@Mapping` é usada para definir mapeamentos específicos entre os campos de origem e destino, especialmente quando os nomes ou tipos diferem.

Java

```java
@Mapper
public interface UsuarioMapper {
    @Mapping(source = "nomeCompleto", target = "nome")
    UsuarioDTO toUsuarioDTO(Usuario usuario);
}
```

Nesse caso, o campo `nomeCompleto` do objeto de origem será mapeado para o campo nome do objeto de destino.

@Mappings

Quando há múltiplos mapeamentos personalizados, a anotação @Mappings agrupa várias anotações @Mapping. Veja:

```java
@Mapper
public interface UsuarioMapper {
    @Mappings({
        @Mapping(source = "nomeCompleto", target = "nome"),
        @Mapping(source = "idadeAtual", target = "idade")
    })
    UsuarioDTO toUsuarioDTO(Usuario usuario);
}
```

@MappingTarget

Essa anotação é usada para atualizar uma instância existente do objeto de destino em vez de criar um novo. É útil, por exemplo, ao trabalhar com objetos gerenciados por frameworks como JPA.

```java
@Mapper
public interface UsuarioMapper {
    void updateUsuarioFromDTO(UsuarioDTO dto, @MappingTarget Usuario
usuario);
}
```

@InheritInverseConfiguration

Permite reutilizar as configurações de mapeamento de um método existente para gerar o mapeamento inverso. Isso reduz redundância no código.

```java
@Mapper
public interface UsuarioMapper {
    UsuarioDTO toUsuarioDTO(Usuario usuario);

    @InheritInverseConfiguration
    Usuario toUsuario(UsuarioDTO dto);
}
```

Vamos reunir essas anotações e outras numa tabela-resumo.

Componente	Descrição
@Mapper	Define uma interface como mapeador.
@Mapping	Especifica regras de mapeamento entre propriedades.
@Mappings	Agrupa múltiplos mapeamentos personalizados.
@MappingTarget	Atualiza um objeto existente em vez de criar um novo.
@InheritInverseConfiguration	Reutiliza mapeamentos para criar a configuração inversa.
@BeanMapping	Oferece controle avançado sobre mapeamentos, como ignorar nulos.
Expressões/Constantes	Permite lógica customizada ou uso de valores fixos durante o mapeamento.

(Inédita/Prof. Felipe Mathias) Assinale a alternativa que representa uma anotação usada para especificar regras de mapeamento entre propriedades, no contexto do Map Struct.

a) @Mapper
b) @Mapping
c) @Mappings
d) @MappingTarget
e) @BeanMapping

Comentários:

A anotação que define regras específicas é a @Mapping. *(Gabarito: Letra B)*

QUESTÕES COMENTADAS

01. (Inédita/Prof. Felipe Mathias) Julgue o item subsecutiva, acerca dos conhecimentos sobre o MapStruct.

O MapStruct é uma biblioteca que realiza mapeamentos entre objetos utilizando reflexão em tempo de execução.

Comentários:

O MapStruct não utiliza reflexão. Ele gera o código de mapeamento em tempo de compilação, resultando em maior performance e evitando sobrecarga em tempo de execução.

Gabarito: Errado

---

02. (Inédita/Prof. Felipe Mathias) Julgue o item subsecutiva, acerca dos conhecimentos sobre o MapStruct.

A anotação @Mapper é usada para marcar uma interface ou classe abstrata como um mapeador no MapStruct.

Comentários:

A anotação @Mapper é o ponto de entrada do MapStruct. Ela identifica que a interface ou classe abstrata será usada para definir mapeamentos, e o código correspondente será gerado automaticamente.

Gabarito: Certo

---

03. (Inédita/Prof. Felipe Mathias) Julgue o item subsecutiva, acerca dos conhecimentos sobre o MapStruct.

A anotação @Mapping permite configurar o mapeamento de propriedades com nomes diferentes entre objetos.

Comentários:

A anotação @Mapping é usada para especificar que uma propriedade de origem com um nome diferente deve ser mapeada para uma propriedade de destino. Por exemplo, @Mapping(source = "nomeCompleto", target = "nome").

Gabarito: Certo

---

04. (Inédita/Prof. Felipe Mathias) Julgue o item subsecutiva, acerca dos conhecimentos sobre o MapStruct.

O MapStruct suporta o mapeamento de coleções, como listas e mapas.

Comentários:

O MapStruct suporta mapeamento automático de coleções (como listas e conjuntos) e mapas, desde que os elementos possam ser mapeados entre si.

Gabarito: Certo

---

05. (Inédita/Prof. Felipe Mathias) Julgue o item subsecutiva, acerca dos conhecimentos sobre o MapStruct.

A anotação `@InheritInverseConfiguration` permite configurar mapeamentos bidirecionais reutilizando a configuração de um método existente.

Comentários:

A anotação @InheritInverseConfiguration permite reaproveitar a lógica de mapeamento de um método para criar a configuração inversa, reduzindo redundância.

Gabarito: Certo

---

06. (Inédita/Prof. Felipe Mathias) Julgue o item subsecutiva, acerca dos conhecimentos sobre o MapStruct.

A anotação @MappingTarget é usada para criar um novo objeto de destino em vez de atualizar um existente.

Comentários:

A anotação @MappingTarget é usada para atualizar um objeto existente em vez de criar um novo, tornando-a útil em cenários como atualização de entidades gerenciadas.

Gabarito: Errado

---

07. (Inédita/Prof. Felipe Mathias) Julgue o item subsecutiva, acerca dos conhecimentos sobre o MapStruct.

O MapStruct não suporta integração com frameworks de injeção de dependência, como Spring ou CDI.

Comentários:

O MapStruct suporta integração com frameworks como Spring e CDI. Isso pode ser configurado usando a propriedade componentModel na anotação @Mapper.

Gabarito: Errado

---

08. (Inédita/Prof. Felipe Mathias) Julgue o item subsecutiva, acerca dos conhecimentos sobre o MapStruct.

O MapStruct permite o uso de expressões customizadas para lógica de conversão em propriedades.

Comentários:

O MapStruct permite definir expressões customizadas na anotação @Mapping usando a propriedade expression, útil para conversões específicas.

Gabarito: Certo

---

09. (Inédita/Prof. Felipe Mathias) Julgue o item subsecutiva, acerca dos conhecimentos sobre o MapStruct.

A configuração de nullValuePropertyMappingStrategy pode ser usada para ignorar campos nulos durante o mapeamento.

Comentários:

A propriedade nullValuePropertyMappingStrategy permite configurar como os campos nulos devem ser tratados, sendo comum usá-la para ignorar a cópia de valores nulos.

Gabarito: Certo

---

10. (Inédita/Prof. Felipe Mathias) Julgue o item subsecutiva, acerca dos conhecimentos sobre o MapStruct.

O MapStruct utiliza bibliotecas externas, como Lombok, para gerar as implementações de mapeadores.

Comentários:

O MapStruct gera as implementações dos mapeadores diretamente em tempo de compilação, sem depender de bibliotecas como Lombok. No entanto, ele pode ser integrado ao Lombok se necessário.

Gabarito: Errado

LISTA DE QUESTÕES

01. (Inédita/Prof. Felipe Mathias) Julgue o item subsecutiva, acerca dos conhecimentos sobre o MapStruct.

O MapStruct é uma biblioteca que realiza mapeamentos entre objetos utilizando reflexão em tempo de execução.

02. (Inédita/Prof. Felipe Mathias) Julgue o item subsecutiva, acerca dos conhecimentos sobre o MapStruct.

A anotação @Mapper é usada para marcar uma interface ou classe abstrata como um mapeador no MapStruct.

03. (Inédita/Prof. Felipe Mathias) Julgue o item subsecutiva, acerca dos conhecimentos sobre o MapStruct.

A anotação @Mapping permite configurar o mapeamento de propriedades com nomes diferentes entre objetos.

04. (Inédita/Prof. Felipe Mathias) Julgue o item subsecutiva, acerca dos conhecimentos sobre o MapStruct.

O MapStruct suporta o mapeamento de coleções, como listas e mapas.

05. (Inédita/Prof. Felipe Mathias) Julgue o item subsecutiva, acerca dos conhecimentos sobre o MapStruct.

A anotação `@InheritInverseConfiguration` permite configurar mapeamentos bidirecionais reutilizando a configuração de um método existente.

06. (Inédita/Prof. Felipe Mathias) Julgue o item subsecutiva, acerca dos conhecimentos sobre o MapStruct.

A anotação @MappingTarget é usada para criar um novo objeto de destino em vez de atualizar um existente.

07. (Inédita/Prof. Felipe Mathias) Julgue o item subsecutiva, acerca dos conhecimentos sobre o MapStruct.

O MapStruct não suporta integração com frameworks de injeção de dependência, como Spring ou CDI.

08. (Inédita/Prof. Felipe Mathias) Julgue o item subsecutiva, acerca dos conhecimentos sobre o MapStruct.

O MapStruct permite o uso de expressões customizadas para lógica de conversão em propriedades.

09. (Inédita/Prof. Felipe Mathias) Julgue o item subsecutiva, acerca dos conhecimentos sobre o MapStruct.

A configuração de nullValuePropertyMappingStrategy pode ser usada para ignorar campos nulos durante o mapeamento.

10. (Inédita/Prof. Felipe Mathias) Julgue o item subsecutiva, acerca dos conhecimentos sobre o MapStruct.

O MapStruct utiliza bibliotecas externas, como Lombok, para gerar as implementações de mapeadores.

GABARITO

GABARITO

1. Errado
2. Certo
3. Certo
4. Certo
5. Certo
6. Errado
7. Errado
8. Certo
9. Certo
10. Errado

SWAGGER

Conceitos Gerais

Quando pensamos no desenvolvimento de APIs, principalmente APIs RESTful, uma das maiores dificuldades enfrentadas por equipes de desenvolvimento é garantir a clareza e a uniformidade na comunicação entre quem desenvolve e quem consome as APIs. Neste cenário, a documentação desempenha um papel crucial, e o **Swagger** surge como uma solução poderosa e amplamente utilizada para **documentar, testar e padronizar APIs**.

Swagger é, na verdade, um ecossistema de ferramentas que facilita o trabalho com APIs, baseado em um padrão internacional chamado OpenAPI Specification (OAS). Antes de detalharmos o funcionamento e as ferramentas do Swagger, é importante entender o impacto que ele tem no ciclo de vida do desenvolvimento de APIs.

Imagine um projeto típico onde desenvolvemos uma API para gerenciar um sistema bancário. Os consumidores dessa API precisam saber quais endpoints estão disponíveis, como interagir com eles (métodos HTTP e parâmetros), quais dados são esperados no corpo da requisição e quais respostas podem ser retornadas. Sem uma boa documentação, essa comunicação frequentemente falha, gerando retrabalho e frustrações. Swagger resolve isso criando uma **documentação automatizada, padronizada e interativa** que permite tanto a leitura da estrutura da API quanto a execução de testes diretamente na interface gerada.

A principal vantagem do Swagger é que ele padroniza o desenvolvimento de APIs, desde o design inicial até os testes finais, promovendo a colaboração entre desenvolvedores, testers e analistas. Com ele, as equipes podem adotar práticas como Design-First ou Code-First.

API, ou **Application Programming Interface**, é uma interface que permite que sistemas diferentes se comuniquem. Uma API RESTful é baseada nos princípios REST (Representational State Transfer), que utiliza os métodos HTTP (GET, POST, PUT, DELETE) para manipular recursos.

Por exemplo, uma API para gerenciar produtos em um e-commerce poderia oferecer endpoints como:

- GET /produtos: para listar todos os produtos.
- POST /produtos: para criar um novo produto.
- PUT /produtos/{id}: para atualizar um produto existente.
- DELETE /produtos/{id}: para remover um produto.

Swagger começou como uma ferramenta para descrever e documentar APIs RESTful. Com o tempo, tornou-se um conjunto de ferramentas integradas, incluindo o Swagger Editor, Swagger UI e Swagger Codegen, todas baseadas na OpenAPI Specification (OAS).

A OpenAPI Specification é uma especificação formal que descreve todos os aspectos de uma API, incluindo:

- Quais endpoints estão disponíveis.
- Quais métodos HTTP eles suportam.
- Quais parâmetros são necessários (e opcionais).
- O formato dos dados esperados no corpo da requisição e na resposta.

Com Swagger, essa descrição não é apenas texto estático; ela é interativa e executável. A partir de um arquivo YAML ou JSON que segue a OAS, o Swagger UI cria uma interface gráfica que permite aos desenvolvedores explorar e testar a API.

É importante entender a evolução do termo "Swagger". Originalmente, Swagger era o nome do framework que introduziu essa abordagem inovadora para documentação de APIs. No entanto, em 2015, a especificação foi renomeada para OpenAPI Specification e passou a ser gerenciada pela OpenAPI Initiative. Swagger, hoje, é um subconjunto de ferramentas que implementam a OAS.

Em outras palavras:

- OpenAPI Specification (OAS): É o padrão que define como uma API deve ser descrita.
- Swagger: É um conjunto de ferramentas que usa o padrão OAS para criar, visualizar e testar APIs.

(CEBRASPE/INPI/2024) Julgue o próximo item, relativo a Hibernate Envers e Swagger.

Um documento Swagger é o equivalente a API REST de um documento WSDL para um serviço web baseado em SOAP.

Comentários:

O WSDL, ou Web Service Description Language, e uma linguagem baseada em XML para descrever serviços baseados em SOAP – basicamente, a sua documentação. E o Swagger age da mesma forma, só que com APIs REST. Portanto, correto o item. *(Gabarito: Certo)*

Componentes

O Swagger é composto por um conjunto robusto de ferramentas que trabalham em conjunto para facilitar o design, a documentação e a interação com APIs RESTful. Esses componentes desempenham papéis específicos no ciclo de vida do desenvolvimento de APIs, e juntos criam um ambiente integrado para desenvolvedores e equipes.

Vamos explorar os principais componentes: Swagger Editor, Swagger UI, e Swagger Codegen, detalhando como cada um contribui para o desenvolvimento eficiente de APIs.

Swagger Editor

O **Swagger Editor** é uma ferramenta que permite **criar, editar e validar especificações OpenAPI** de forma visual e intuitiva. Ele é essencial para definir todos os detalhes de uma API, desde seus endpoints e métodos até parâmetros, respostas e esquemas de dados. O editor suporta os formatos YAML e JSON, que são utilizados para descrever APIs de acordo com a OpenAPI Specification (OAS).

O editor oferece uma interface de texto onde podemos escrever ou colar o arquivo OpenAPI. Conforme escrevemos, ele exibe erros de validação, garantindo que seguimos a sintaxe correta. Ao lado da área de edição, o Swagger Editor fornece uma visualização prévia que simula como a API será exibida no Swagger UI.

O Swagger Editor é ideal tanto para abordagens **Design-First** (onde a API é planejada antes de sua implementação) quanto **Code-First** (onde a documentação é criada a partir do código já existente).

Swagger UI

O **Swagger UI** é o componente que **transforma a especificação OpenAPI em uma interface gráfica** interativa. Ele é amplamente utilizado para disponibilizar a documentação de APIs de forma acessível e prática, permitindo que desenvolvedores, testadores e usuários finais explorem e testem os endpoints diretamente no navegador.

Um dos grandes diferenciais do Swagger UI é a funcionalidade **"Try it out"**, que permite executar requisições diretamente na interface, enviando parâmetros e visualizando respostas em tempo real. Imagine um o endpoint /usuarios descrito no Swagger Editor. No Swagger UI, ele será exibido como:

- Método: GET
- Descrição: "Retorna a lista de usuários."
- Resposta esperada: JSON com uma lista de objetos contendo id, nome e email.

Ao clicar em "Try it out", o Swagger UI exibirá um formulário onde podemos inserir os parâmetros necessários (se houver) e executar a requisição, exibindo a resposta retornada pela API diretamente na interface.

Veja como o componente costuma aparecer em provas:

(FGV/TJ RN/2023) A equipe de análise e desenvolvimento de sistemas do TJRN está implementando uma nova Application Programming Interface (API) com o apoio de ferramentas Swagger. Para explorar a

especificação da API de forma visual, a equipe utiliza a ferramenta Swagger, que é capaz de gerar, no próprio navegador web, a documentação visual da API, diretamente do documento de especificação. No entanto, a ferramenta utilizada não permite alterar o documento de especificação.

Para explorar visualmente a especificação da API, a equipe utiliza a ferramenta Swagger:

a) UI;
b) Core;
c) Parser;
d) Codegen;
e) Inspector.

Comentários:

A questão quer uma ferramenta do Swagger que "é capaz de gerar, no próprio navegador web, a documentação visual da API". Essa descrição se adequa ao conceito do Swagger UI. *(Gabarito: Letra A)*

Swagger Codegen

O **Swagger Codegen** é uma ferramenta que **gera automaticamente código-fonte baseado na especificação OpenAPI**. Ele suporta diversas linguagens de programação e frameworks, permitindo que desenvolvedores economizem tempo na implementação de clientes e servidores para APIs.

O Swagger Codegen pode criar bibliotecas para consumir APIs em linguagens como Python, Java, C#, Ruby e outras. Isso elimina a necessidade de implementar manualmente as chamadas à API. Ele também pode gerar a estrutura de código para criar servidores que implementem a API descrita no OpenAPI.

Para gerar o código, usamos um código de CLI do Swagger. Por exemplo:

```
swagger-codegen generate -i openapi.yaml -l python -o ./cliente-python
```

Este comando criará uma biblioteca Python que pode ser importada e utilizada para consumir os endpoints da API descrita no arquivo openapi.yaml.

(IDECAN/TJ PI/2022) O Swagger é composto por um conjunto de ferramentas que nos permite: modelar, desenvolver e documentar APIs. O processo de desenvolvimento de uma API exige uma série de definições técnicas como por exemplo: dados recebidos, dados retornados, endpoínts e métodos de autenticação. O

Swagger auxilia o usuário na definição de todas essas questões e facilita a construção e documentação da API que incorporará todas essas características.

A respeito das ferramentas que compõem o Swagger, analise as afirmativas abaixo e marque alternativa correta.

I. O Swagger possui ferramentas como: SwaggerEditor, Swagger UI e Swagger Codegen. Todas essas ferramentas são gratuitas e de código aberto.
II. O Swagger Codegen permite a criação de código fonte para sua API. Ele suporta linguagens como: aspnetcore, PHP, phyton, node, erlang.
III. O SwaggerEditor é ferramenta que permite a definição dos contratos e outras características que deverão existir em nossa API. Essa ferramenta nos permite salvar as definições da API em diferentes formatos como: JSON, YAML e HTML.

a) Apenas as afirmativas I e II estão corretas.
b) Apenas as afirmativas I e III estão corretas.
c) Apenas a afirmativa I está correta.
d) Apenas as afirmativas II e III estão corretas.
e) Todas as afirmativas estão corretas.

Comentários:

Vamos analisar os itens.

I. Certo. De fato, esses são exemplos de componentes que englobam a suíte do Swagger, sendo todas as citadas de código aberto e gratuitas.
II. Certo. O Codegen gera códigos automaticamente para uma diversidade de linguagens, incluindo as citadas.
III. Errado. O Swagger Editor gera arquivos JSON e YAML, mas não é possível salvar as definições em HTML.

Portanto, corretos os itens I e II. *(Gabarito: Letra A)*

Outros Componentes

Existem alguns outros componentes, que não são tão protagonistas quanto os três que acabamos de ver. Vamos reunir todos, incluindo os que já falamos, numa tabela-resumo.

Componente	Descrição

Swagger Codegen	Ferramenta para gerar código-fonte automaticamente a partir de uma especificação OpenAPI. Suporta várias linguagens como Java, Python, C#, Ruby, entre outras.
Swagger UI	Interface gráfica que transforma a especificação OpenAPI em uma documentação interativa, permitindo explorar e testar os endpoints diretamente no navegador.
Swagger Editor	Ambiente visual para criar, editar e validar especificações OpenAPI em formato YAML ou JSON, com suporte a feedback em tempo real e visualização integrada.
Swagger JS	Biblioteca em JavaScript que permite lidar com especificações OpenAPI, oferecendo suporte para análise, validação e execução de chamadas API diretamente no ambiente web.
Swagger Node	Framework para desenvolvimento de APIs baseado em Node.js, integrando com OpenAPI para geração de rotas, validação de dados e documentação.
Swagger Socket	Ferramenta para APIs em tempo real, permitindo criar conexões bidirecionais, expor e invocar definições de APIs baseadas em WebSockets, otimizando comunicação em tempo real entre cliente e servidor.
Swagger Parser	Biblioteca para validar e analisar especificações OpenAPI, permitindo leitura e manipulação programática de arquivos YAML ou JSON.
Swagger Hub	Plataforma colaborativa baseada em nuvem desenvolvida para facilitar o design, documentação e gerenciamento de APIs utilizando a OpenAPI Specification (OAS). Ele combina as funcionalidades do Swagger Editor e do Swagger UI em um ambiente centralizado, permitindo que equipes colaborem no desenvolvimento de APIs de maneira eficiente e organizada.

(CEBRASPE/TST/2024) Assinale a opção que apresenta a ferramenta utilizada no Swagger para expor e invocar definições de APIs feitas com o próprio Swagger.

a) Swagger Socket
b) Swagger Editor
c) Swagger Parse
d) Swagger Codegen
e) Swagger JS

Comentários:

Para *expormos* e *invocarmos* as definições de APIs feitas com o Swagger, usamos o Swagger Socket. *(Gabarito: Letra A)*

Comandos CLI

Sempre que temos uma ferramenta com comandos CLI, é interessante sabermos quais comandos existem para nos blindarmos de eventuais cobranças.

Comando	Descrição
swagger project create	Cria um novo projeto Swagger baseado em uma especificação OpenAPI.
swagger project start	Inicia o servidor local para testar o projeto Swagger.
swagger project edit	Abre o Swagger Editor em uma interface web para editar a especificação do projeto.
swagger project test	Executa os testes definidos para a API no projeto.
swagger project validate	Valida a especificação OpenAPI para verificar erros ou inconsistências.
swagger generate server	Gera a estrutura de código do lado servidor baseada na especificação OpenAPI, permitindo a implementação dos endpoints.
swagger generate client	Gera código do cliente para consumir uma API com base na especificação OpenAPI.
swagger generate docs	Gera a documentação da API em formato estático (HTML) a partir da especificação OpenAPI.
swagger mock	Inicia um servidor de mock (simulação) para testar a API sem precisar de uma implementação real.
swagger convert	Converte especificações entre diferentes versões da OpenAPI (ex.: de OpenAPI 2.0 para OpenAPI 3.0).
swagger help	Exibe ajuda sobre os comandos disponíveis na CLI do Swagger.
swagger version	Exibe a versão instalada do Swagger CLI.

Integração Swagger-Spring (Java)

O Swagger se integra ao Spring Framework de maneira eficiente, facilitando a documentação de APIs desenvolvidas com Spring Boot. A integração **é baseada no uso de anotações diretamente no código** para descrever endpoints e outros aspectos da API. Com a biblioteca Springfox, podemos gerar automaticamente uma especificação OpenAPI e exibir essa documentação em uma interface Swagger UI.

Para implementarmos as dependências, é comum usarmos o Springfox, uma biblioteca amplamente utilizada que objetiva fazer implementações do Swagger em Projetos Spring:

```
<dependency>
    <groupId>io.springfox</groupId>
```

```
        <artifactId>springfox-boot-starter</artifactId>
        <version>3.0.0</version>
    </dependency>
```

As anotações são o principal meio de integrar informações do Swagger diretamente no código Spring. Elas são usadas para documentar endpoints, parâmetros, respostas e modelos de dados. As principais anotações são:

Anotações	Descrição
@EnableSwagger	Anotação usada em versões antigas do Springfox para habilitar o Swagger na aplicação. Descontinuada em versões recentes.
@Api	Marca uma classe como um recurso documentável no Swagger, permitindo agrupar e descrever endpoints.
@ApiOperation	Define detalhes de uma operação (endpoint), como sua funcionalidade e notas explicativas.
@ApiParam	Documenta os parâmetros de entrada de um endpoint, como descrição e se é obrigatório.
@ApiResponse	Documenta as possíveis respostas de um endpoint, incluindo códigos HTTP e descrições.
@Schema	Documenta os atributos de um modelo de dados, permitindo definir descrições, exemplos e restrições.

(FCC/TRT 19/2022) Para documentar uma aplicação Spring Boot com Swagger2 é necessário ativar o Swagger na classe SwaggerConfig usando a anotação

a) @Swagger2ApiOn
b) @EnableSwagger2Config
c) @Swagger2Config
d) @Swagger2Application
e) @EnableSwagger2

Comentários:

A questão cobra uma abordagem deprecada, usada em versões mais antigas do Swagger – como o Swagger2. Nesse caso, a anotação a ser utilizada na SwaggerConfig é a @EnableSwagger2. *(Gabarito: Letra E)*

QUESTÕES COMENTADAS

01. (CEBRASPE/STJ/2024) Em relação à linguagem de programação Java, à arquitetura distribuída de microsserviços e à biblioteca Flyway, julgue o próximo item.

Swagger é uma interface API compatível com Java que permite que dois sistemas computacionais troquem informações com ausência de estado.

Comentários:

A troca de informações com ausência de estado é uma descrição das APIs REST. Embora Swagger seja compatível com Java, ele não é uma interface API, e sim uma forma de criarmos especificações e documentações (e as próprias APIs), não uma interface API em si.

Gabarito: Errado

---

02. (IBFC/TRF 5/2024) APIs (Application Programming Interface) são amplamente utilizadas para comunicação entre sistemas. Assinale a alternativa que apresenta corretamente o propósito e o uso do Swagger no desenvolvimento de APIs.

a) Swagger é uma ferramenta exclusiva para testar APIs, sem suporte para design ou documentação
b) Swagger é um software apenas disponível comercialmente, destinado apenas a engenheiros técnicos
c) Swagger foi criado pela SoapUI, uma empresa especializada em desenvolvimento de APIs
d) Swagger é uma suíte de ferramentas que cobre todo o ciclo de vida de uma API, incluindo design, documentação, teste e implantação

Comentários:

Vamos analisar as alternativas.

a) Errado. Embora o Swagger possa ser usado para testar APIs, ele não é limitado a isso. Ele aborda todo o ciclo de vida, especialmente nas fases de documentação e *design*.
b) Errado. Swagger é *open source* e gratuito.
c) Errado. O Swagger foi criado por uma empresa denominada Reverb Technologies.
d) Certo. Conforme vimos na letra A.

Portanto, correta a letra D.

Gabarito: Letra D

---

03. (IBFC/TRF 5/2024) O desenvolvimento de APIs vem se tornando cada vez mais frequente, devido a necessidade da comunicação entre sistemas, desta forma muitas tecnologias foram e estão sendo criadas para apoiar os desenvolvedores a terem melhor produtividade e organização ao longo do desenvolvimento das aplicações. O swagger no contexto do desenvolvimento de APIs é:

a) um banco de dados exclusivo para armazenar informações de APIs
b) uma linguagem de programação para criar APIs
c) um servidor web para hospedar APIs
d) uma ferramenta para projetar, documentar e consumir APIs RESTful

Comentários:

O Swagger é uma ferramenta que "gerencia" as diversas fases do ciclo de vida de APIs REST, desde a criação até os testes e implementação, com um foco nas fases de documentação e *design* da API. Portanto, correta a letra D.

Gabarito: Letra D

---

04. (IBFC/TRF 5/2024) Assinale a alternativa que preencha corretamente a lacuna:

Todo e qualquer software considerando boas práticas de desenvolvimento, pressupõe-se ser documentado. Ao se tratar de APIs, estas requerem documentações imprescindíveis, para que um outro desenvolvedor possa compreender informações como: arquitetura de integração, dados a serem enviados/consumidos, entre outras informações técnicas. Desta forma, o Swagger é uma ferramenta que auxilia este processo de compreensão sobre a API que será fornecida. O Swagger contribui para o desenvolvimento de APIs Restful _______.

a) permitindo a documentação do sistema operacional, versões dos editores de código (IDEs) e demais informações
b) permitindo a documentação padronizada e interativa das APIs
c) facilitando a geração automática de código-fonte para APIs e suas estruturas de bancos de dados
d) oferecendo suporte exclusivo para APIs SOAP

Comentários:

Vamos ver qual alternativa se alinha melhor om a lacuna.

a) Errado. O Swagger foca na documentação das APIs REST, não em sistemas operacionais, IDEs e etc.
b) Certo. De fato, o Swagger foca na documentação padronizada e interativa (a partir do Swagger UI) para APIs REST.
c) Errado. Para códigos fonte de APIs temos, de fato, o Swagger Codegen. Mas o Swagger não faz a geração das estruturas de bancos de dados.
d) Errado. O Swagger trabalha em APIs REST. Para APIs SOAP, usamos o WSDL.

Portanto, correta a letra B.

Gabarito: Letra B

---

05. (CEBRASPE/TST/2024) Assinale a opção que apresenta a ferramenta utilizada no Swagger para expor e invocar definições de APIs feitas com o próprio Swagger.

a) Swagger Socket
b) Swagger Editor
c) Swagger Parse
d) Swagger Codegen
e) Swagger JS

Comentários:

Para *expormos* e *invocarmos* as definições de APIs feitas com o Swagger, usamos o Swagger Socket.

Gabarito: Letra A

---

06. (CEBRASPE/INPI/2024) Julgue o próximo item, relativo a Hibernate Envers e Swagger.

Um documento Swagger é o equivalente a API REST de um documento WSDL para um serviço web baseado em SOAP.

Comentários:

O WSDL, ou Web Service Description Language, e uma linguagem baseada em XML para descrever serviços baseados em SOAP – basicamente, a sua documentação. E o Swagger age da mesma forma, só que com APIs REST. Portanto, correto o item.

Gabarito: Certo

---

07. (FCC/TRT 12/2023) Uma Analista está desenvolvendo uma API REST para um aplicativo e deseja criar a documentação usando o Swagger. Essa documentação é criada adequadamente através

a) do Swagger Codegen, que permite a criação manual da especificação OpenAPI.
b) da interface Swagger Editor, que permite criar a especificação OpenAPI interativamente.
c) da elaboração de um arquivo de documentação PDF com o Swagger PDF Editor contendo a descrição das APIs.
d) da interface Swagger GetJet, que cria e exibe automaticamente a documentação.
e) da elaboração de um arquivo XML de forma manual, contendo a especificação OpenDocumentAPI.

Comentários:

Vamos analisar as alternativas.

a) Errado. O Swagger Codegen não é usado para criar especificações OpenAPI. Ele é utilizado para gerar código-fonte (clientes ou servidores) com base em uma especificação já existente.
b) Certo. É o Swagger Editor que permite criar e especificar a API através de um trabalho manual e interativo.

c) Errado. Não temos um “Swagger PDF Editor”. A documentação gerada pelo Swagger é interativa e baseada em OpenAPI, geralmente visualizada via Swagger UI ou arquivos YAML/JSON.
d) Errado. Não existe uma ferramenta chamada "Swagger GetJet". A documentação automática é gerada por ferramentas como Swagger UI, mas depende de uma especificação OpenAPI criada previamente.
e) Errado. A OpenAPI Specification utiliza os formatos YAML ou JSON, não XML. Além disso, "OpenDocumentAPI" não é relacionado ao Swagger ou OpenAPI.

Sendo assim, correta a letra B.

Gabarito: Letra B

---

08. (FGV/TJ SE/2023) O time de desenvolvimento de sistemas (TDS) tem utilizado o Swagger conjugado ao desenvolvimento de API Restful. Utilizando o Swagger:

a) as requisições PUT requerem autorização além da autenticação;
b) o grupo de testes pode testar os serviços da API sem a camada de frontend;
c) está dispensada a autenticação à API pois não há riscos de injeção de código;
d) as respostas aos serviços com conteúdos JSON não são exibidas, embora sejam exibidos os códigos de retorno;
e) os códigos de retorno de erro devem obrigatoriamente ser definidos e os códigos de sucesso são opcionais

Comentários:

Vamos analisar cada alternativa.

a) Errado. O Swagger não impõe ou gerencia autenticação ou autorização. Ele apenas documenta e, se configurado, exibe informações sobre os requisitos de segurança de uma API (como autenticação via tokens ou headers). A necessidade de autorização ou autenticação depende da implementação da API, não do Swagger.
b) Certo. O Swagger fornece ferramentas como o Swagger UI, que permite que os desenvolvedores e testadores interajam diretamente com a API RESTful sem depender de um frontend. Por meio de uma interface gráfica, é possível enviar requisições, testar endpoints, visualizar parâmetros e analisar respostas (incluindo conteúdos JSON), tudo de forma independente da aplicação cliente.
c) Errado. O uso do Swagger não dispensa a necessidade de autenticação na API. Além disso, APIs RESTful sempre podem estar sujeitas a riscos como injeção de código ou outros ataques se não forem implementadas com práticas seguras. O Swagger apenas documenta os endpoints; ele não oferece proteção direta.
d) Errado. O Swagger UI exibe tanto os códigos de retorno quanto os conteúdos das respostas (como JSON), permitindo uma visão completa da interação com os endpoints.
e) Errado. Não há obrigatoriedade em definir códigos de erro ou de sucesso no Swagger. Embora seja uma boa prática documentar ambos, isso depende exclusivamente do desenvolvedor da API.

Portanto, correta a letra B.

Gabarito: Letra B

---

09. (FGV/TJ RN/2023) A equipe de análise e desenvolvimento de sistemas do TJRN está implementando uma nova Application Programming Interface (API) com o apoio de ferramentas Swagger. Para explorar a especificação da API de forma visual, a equipe utiliza a ferramenta Swagger, que é capaz de gerar, no próprio navegador web, a documentação visual da API, diretamente do documento de especificação. No entanto, a ferramenta utilizada não permite alterar o documento de especificação.

Para explorar visualmente a especificação da API, a equipe utiliza a ferramenta Swagger:

a) UI;
b) Core;
c) Parser;
d) Codegen;
e) Inspector.

Comentários:

Queremos "explorar a especificação da API de forma visual". Da suíte Swagger, a ferramenta responsável por isso é a Swagger UI.

Gabarito: Letra A

---

10. (IBFC/TJ MG/2022) Quanto à funcionalidade específica do Swagger, assinale a alternativa correta.

a) É um padrão de arquitetura de software arquitetural criado por Robert Martin, que favorece a implementação de sistemas com reusabilidade de código, coesão, independência de tecnologia e testabilidade
b) É uma plataforma de prototipagem e de simulação de software livre para a criação de contêineres
c) É a combinação de diversas ferramentas em uma única interface gráfica para o desenvolvimento de aplicações
d) Ramo da Inteligência Artificial que utiliza modelos analíticos a partir da coleta, análise e correlação de dados
e) Permite o desenvolvimento em todo o ciclo de vida da API, desde o design e a documentação até o teste e a implantação

Comentários:

Vamos analisar as alternativas.

a) Errado. A descrição corresponde aos princípios da Arquitetura Limpa (Clean Architecture), que não tem relação direta com o Swagger.
b) Errado. Essa descrição se aproxima do funcionamento de ferramentas como o Docker, não do Swagger.
c) Errado. Embora o Swagger combine várias ferramentas, ele não é utilizado para o desenvolvimento completo de aplicações, mas para o ciclo de vida e documentação de APIs.
d) Errado. A descrição é relacionada a Machine Learning ou Data Analytics, que não são funcionalidades do Swagger.

e) Certo. O Swagger é um conjunto de ferramentas que oferece suporte ao desenvolvimento de APIs RESTful em todas as etapas de seu ciclo de vida, incluindo design, documentação, testes e implantação.

Correta a letra E, portanto.

Gabarito: Letra E

---

11. (FCC/TRT 23/2022) No componente controller de uma aplicação criada com recursos Spring Boot foram incluídas as anotações abaixo a um método REST.

```
@ApiOperation(value = "Obter a lista de processos no sistema", response =
Iterable.class, tags
= "getProcessos")
@ApiResponses(value = {
      @ApiResponse(code = 200, message = "Successo|OK"),
      @ApiResponse(code = 401, message = "Não autorizado"),
      @ApiResponse(code = 403, message = "Proibido"),
      @ApiResponse(code = 404, message = "Não encontrado") })
@RequestMapping(value = "/getProcessos")
public List<Processos> getProcessos() {
    return processos;
}
```

`@ApiOperation` e `@ApiResponses` são anotações

a) MapStruct.
b) Hibernate.
c) Spring Eureka.
d) Spring Zuul.
e) Swagger.

Comentários:

@ApiOperation e @ApiResponses são anotações do Swagger. A @ApiOperation define detalhes de uma operação (endpoint), como sua funcionalidade e notas explicativas; já a @ApiResponses agrupa um conjunto de @ApiResponse, que documenta as possíveis respostas de um endpoint, incluindo códigos HTTP e descrições.

Gabarito: Letra E

---

12. (IDECAN/TJ PI/2022) O Swagger é composto por um conjunto de ferramentas que nos permite: modelar, desenvolver e documentar APIs. O processo de desenvolvimento de uma API exige uma série de definições técnicas como por exemplo: dados recebidos, dados retornados, endpoínts e métodos de autenticação. O

Swagger auxilia o usuário na definição de todas essas questões e facilita a construção e documentação da API que incorporará todas essas características.

A respeito das ferramentas que compõem o Swagger, analise as afirmativas abaixo e marque alternativa correta.

> I. O Swagger possui ferramentas como: SwaggerEditor, Swagger UI e Swagger Codegen. Todas essas ferramentas são gratuitas e de código aberto.
> II. O Swagger Codegen permite a criação de código fonte para sua API. Ele suporta linguagens como: aspnetcore, PHP, phyton, node, erlang.
> III. O SwaggerEditor é ferramenta que permite a definição dos contratos e outras características que deverão existir em nossa API. Essa ferramenta nos permite salvar as definições da API em diferentes formatos como: JSON, YAML e HTML.

a) Apenas as afirmativas I e II estão corretas.
b) Apenas as afirmativas I e III estão corretas.
c) Apenas a afirmativa I está correta.
d) Apenas as afirmativas II e III estão corretas.
e) Todas as afirmativas estão corretas.

Comentários:

Vamos analisar os itens.

> I. Certo. De fato, esses são exemplos de componentes que englobam a suíte do Swagger, sendo todas as citadas de código aberto e gratuitas.
> II. Certo. O Codegen gera códigos automaticamente para uma diversidade de linguagens, incluindo as citadas.
> III. Errado. O Swagger Editor gera arquivos JSON e YAML, mas não é possível salvar as definições em HTML.

Portanto, corretos os itens I e II.

Gabarito: Letra A

---

13. (CEBRASPE/TST/2024) Assinale a opção que apresenta a ferramenta utilizada no Swagger para expor e invocar definições de APIs feitas com o próprio Swagger.

a) Swagger Socket
b) Swagger Editor
c) Swagger Parse
d) Swagger Codegen
e) Swagger JS

Comentários:

Para *expormos* e *invocarmos* as definições de APIs feitas com o Swagger, usamos o Swagger Socket.

Gabarito: Letra A

---

14. (FCC/TRT 4/2022) Para documentar uma API acessível externamente a partir de um cenário de microsserviços, um Analista utilizou a especificação Swagger. Para cada operação RESTful na API, ele adicionou uma anotação A, juntamente com anotações B no método Java correspondente, para descrever a operação e suas respostas de erro esperadas. As anotações A e B são, respectivamente,

a) @describeOperation e @errorResponse.
b) @restOperation e @restResponse.
c) @getInfoOperation e @getErrorResponse.
d) @ApiOperation e @ApiResponse.
e) @swaggerOperation e @swaggerResponse.

Comentários:

Para descrever a operação e as respostas de erro esperadas, as anotações a serem usadas são, respectivamente, @ApiOpeartion e @ApiResponse.

Gabarito: Letra D

---

15. (FCC/TRT 4/2022) Em um código Swagger escrito em JSON, a URL Base é formada por

a) host, baseForms e schemes.
b) consumes, host e produces.
c) host, basePath e schemes.
d) produces, schemes e baseForms.
e) application, get e basePath.

Comentários:

Em um código Swagger (baseado na especificação OpenAPI 2.0), a URL Base de uma API é definida pelos seguintes campos:

- host: Representa o domínio ou endereço do servidor onde a API está hospedada (ex.: api.exemplo.com).
- basePath: Define o caminho raiz da API (ex.: /v1 ou /api), que é anexado ao host.
- schemes: Indica os esquemas de comunicação suportados, como http ou https.

Portanto, correta a letra C.

Gabarito: Letra C

---

16. (IESES/CREA SC/2022) Com relação ao Swagger, é correto afirmar, EXCETO:

a) É uma aplicação que auxilia os desenvolvedores a definir, criar, documentar e consumir APIs REST.

b) Fornece ferramentas para auxiliar na definição do arquivo de configuração e interagir com API através das definições do arquivo de configuração.
c) É composto de um arquivo de configuração, que pode ser definido em YAML ou JSON.
d) É uma aplicação desenvolvida pela Microsoft que auxilia os desenvolvedores a definir, criar, documentar e consumir APIs REST.

Comentários:

Vamos analisar as alternativas, procurando a incorreta.

a) Certo. O Swagger oferece suporte em todo o ciclo de vida de APIs REST, permitindo definição, documentação, testes e consumo.
b) Certo. Ferramentas como Swagger Editor e Swagger UI ajudam na criação e interação com APIs a partir das especificações OpenAPI.
c) Certo. A especificação OpenAPI utilizada pelo Swagger permite que APIs sejam descritas em arquivos no formato YAML ou JSON.
d) Errado. O Swagger foi originalmente criado pela Reverb Technologies, liderada por Tony Tam, e posteriormente evoluiu para um conjunto de ferramentas baseado no padrão OpenAPI Specification (OAS). Ele não foi desenvolvido pela Microsoft. Hoje, o Swagger é mantido por diferentes organizações e é amplamente utilizado em ambientes multiplataforma, independente de qualquer fornecedor específico, como Microsoft.

Portanto, incorreta a letra D.

Gabarito: Letra D

LISTA DE QUESTÕES

01. (CEBRASPE/STJ/2024) Em relação à linguagem de programação Java, à arquitetura distribuída de microsserviços e à biblioteca Flyway, julgue o próximo item.

Swagger é uma interface API compatível com Java que permite que dois sistemas computacionais troquem informações com ausência de estado.

02. (IBFC/TRF 5/2024) APIs (Application Programming Interface) são amplamente utilizadas para comunicação entre sistemas. Assinale a alternativa que apresenta corretamente o propósito e o uso do Swagger no desenvolvimento de APIs.

a) Swagger é uma ferramenta exclusiva para testar APIs, sem suporte para design ou documentação
b) Swagger é um software apenas disponível comercialmente, destinado apenas a engenheiros técnicos
c) Swagger foi criado pela SoapUI, uma empresa especializada em desenvolvimento de APIs
d) Swagger é uma suíte de ferramentas que cobre todo o ciclo de vida de uma API, incluindo design, documentação, teste e implantação

03. (IBFC/TRF 5/2024) O desenvolvimento de APIs vem se tornando cada vez mais frequente, devido a necessidade da comunicação entre sistemas, desta forma muitas tecnologias foram e estão sendo criadas para apoiar os desenvolvedores a terem melhor produtividade e organização ao longo do desenvolvimento das aplicações. O swagger no contexto do desenvolvimento de APIs é:

a) um banco de dados exclusivo para armazenar informações de APIs
b) uma linguagem de programação para criar APIs
c) um servidor web para hospedar APIs
d) uma ferramenta para projetar, documentar e consumir APIs RESTful

04. (IBFC/TRF 5/2024) Assinale a alternativa que preencha corretamente a lacuna:

Todo e qualquer software considerando boas práticas de desenvolvimento, pressupõe-se ser documentado. Ao se tratar de APIs, estas requerem documentações imprescindíveis, para que um outro desenvolvedor possa compreender informações como: arquitetura de integração, dados a serem enviados/consumidos, entre outras informações técnicas. Desta forma, o Swagger é uma ferramenta que auxilia este processo de compreensão sobre a API que será fornecida. O Swagger contribui para o desenvolvimento de APIs Restful _______.

a) permitindo a documentação do sistema operacional, versões dos editores de código (IDEs) e demais informações
b) permitindo a documentação padronizada e interativa das APIs
c) facilitando a geração automática de código-fonte para APIs e suas estruturas de bancos de dados
d) oferecendo suporte exclusivo para APIs SOAP

05. (CEBRASPE/TST/2024) Assinale a opção que apresenta a ferramenta utilizada no Swagger para expor e invocar definições de APIs feitas com o próprio Swagger.

a) Swagger Socket
b) Swagger Editor
c) Swagger Parse
d) Swagger Codegen
e) Swagger JS

06. (CEBRASPE/INPI/2024) Julgue o próximo item, relativo a Hibernate Envers e Swagger.

Um documento Swagger é o equivalente a API REST de um documento WSDL para um serviço web baseado em SOAP.

07. (FCC/TRT 12/2023) Uma Analista está desenvolvendo uma API REST para um aplicativo e deseja criar a documentação usando o Swagger. Essa documentação é criada adequadamente através

a) do Swagger Codegen, que permite a criação manual da especificação OpenAPI.
b) da interface Swagger Editor, que permite criar a especificação OpenAPI interativamente.
c) da elaboração de um arquivo de documentação PDF com o Swagger PDF Editor contendo a descrição das APIs.
d) da interface Swagger GetJet, que cria e exibe automaticamente a documentação.
e) da elaboração de um arquivo XML de forma manual, contendo a especificação OpenDocumentAPI.

08. (FGV/TJ SE/2023) O time de desenvolvimento de sistemas (TDS) tem utilizado o Swagger conjugado ao desenvolvimento de API Restful. Utilizando o Swagger:

a) as requisições PUT requerem autorização além da autenticação;
b) o grupo de testes pode testar os serviços da API sem a camada de frontend;
c) está dispensada a autenticação à API pois não há riscos de injeção de código;
d) as respostas aos serviços com conteúdos JSON não são exibidas, embora sejam exibidos os códigos de retorno;
e) os códigos de retorno de erro devem obrigatoriamente ser definidos e os códigos de sucesso são opcionais

09. (FGV/TJ RN/2023) A equipe de análise e desenvolvimento de sistemas do TJRN está implementando uma nova Application Programming Interface (API) com o apoio de ferramentas Swagger. Para explorar a especificação da API de forma visual, a equipe utiliza a ferramenta Swagger, que é capaz de gerar, no próprio navegador web, a documentação visual da API, diretamente do documento de especificação. No entanto, a ferramenta utilizada não permite alterar o documento de especificação.

Para explorar visualmente a especificação da API, a equipe utiliza a ferramenta Swagger:

a) UI;
b) Core;
c) Parser;
d) Codegen;

e) Inspector.

10. (IBFC/TJ MG/2022) Quanto à funcionalidade específica do Swagger, assinale a alternativa correta.

a) É um padrão de arquitetura de software arquitetural criado por Robert Martin, que favorece a implementação de sistemas com reusabilidade de código, coesão, independência de tecnologia e testabilidade
b) É uma plataforma de prototipagem e de simulação de software livre para a criação de contêineres
c) É a combinação de diversas ferramentas em uma única interface gráfica para o desenvolvimento de aplicações
d) Ramo da Inteligência Artificial que utiliza modelos analíticos a partir da coleta, análise e correlação de dados
e) Permite o desenvolvimento em todo o ciclo de vida da API, desde o design e a documentação até o teste e a implantação

11. (FCC/TRT 23/2022) No componente controller de uma aplicação criada com recursos Spring Boot foram incluídas as anotações abaixo a um método REST.

```
@ApiOperation(value = "Obter a lista de processos no sistema", response =
Iterable.class, tags
= "getProcessos")
@ApiResponses(value = {
      @ApiResponse(code = 200, message = "Successo|OK"),
      @ApiResponse(code = 401, message = "Não autorizado"),
      @ApiResponse(code = 403, message = "Proibido"),
      @ApiResponse(code = 404, message = "Não encontrado") })
@RequestMapping(value = "/getProcessos")
public List<Processos> getProcessos() {
    return processos;
}
```

`@ApiOperation` e `@ApiResponses` são anotações

a) MapStruct.
b) Hibernate.
c) Spring Eureka.
d) Spring Zuul.
e) Swagger.

12. (IDECAN/TJ PI/2022) O Swagger é composto por um conjunto de ferramentas que nos permite: modelar, desenvolver e documentar APIs. O processo de desenvolvimento de uma API exige uma série de definições técnicas como por exemplo: dados recebidos, dados retornados, endpoints e métodos de autenticação. O Swagger auxilia o usuário na definição de todas essas questões e facilita a construção e documentação da API que incorporará todas essas características.

A respeito das ferramentas que compõem o Swagger, analise as afirmativas abaixo e marque alternativa correta.

I. O Swagger possui ferramentas como: SwaggerEditor, Swagger UI e Swagger Codegen. Todas essas ferramentas são gratuitas e de código aberto.
II. O Swagger Codegen permite a criação de código fonte para sua API. Ele suporta linguagens como: aspnetcore, PHP, phyton, node, erlang.
III. O SwaggerEditor é ferramenta que permite a definição dos contratos e outras características que deverão existir em nossa API. Essa ferramenta nos permite salvar as definições da API em diferentes formatos como: JSON, YAML e HTML.

a) Apenas as afirmativas I e II estão corretas.
b) Apenas as afirmativas I e III estão corretas.
c) Apenas a afirmativa I está correta.
d) Apenas as afirmativas II e III estão corretas.
e) Todas as afirmativas estão corretas.

13. (CEBRASPE/TST/2024) Assinale a opção que apresenta a ferramenta utilizada no Swagger para expor e invocar definições de APIs feitas com o próprio Swagger.

a) Swagger Socket
b) Swagger Editor
c) Swagger Parse
d) Swagger Codegen
e) Swagger JS

14. (FCC/TRT 4/2022) Para documentar uma API acessível externamente a partir de um cenário de microsserviços, um Analista utilizou a especificação Swagger. Para cada operação RESTful na API, ele adicionou uma anotação A, juntamente com anotações B no método Java correspondente, para descrever a operação e suas respostas de erro esperadas. As anotações A e B são, respectivamente,

a) @describeOperation e @errorResponse.
b) @restOperation e @restResponse.
c) @getInfoOperation e @getErrorResponse.
d) @ApiOperation e @ApiResponse.
e) @swaggerOperation e @swaggerResponse.

15. (FCC/TRT 4/2022) Em um código Swagger escrito em JSON, a URL Base é formada por

a) host, baseForms e schemes.
b) consumes, host e produces.
c) host, basePath e schemes.
d) produces, schemes e baseForms.
e) application, get e basePath.

16. (IESES/CREA SC/2022) Com relação ao Swagger, é correto afirmar, EXCETO:

a) É uma aplicação que auxilia os desenvolvedores a definir, criar, documentar e consumir APIs REST.

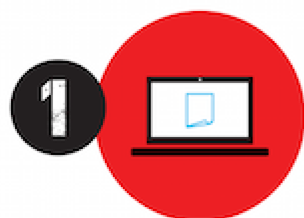
b) Fornece ferramentas para auxiliar na definição do arquivo de configuração e interagir com API através das definições do arquivo de configuração.
c) É composto de um arquivo de configuração, que pode ser definido em YAML ou JSON.
d) É uma aplicação desenvolvida pela Microsoft que auxilia os desenvolvedores a definir, criar, documentar e consumir APIs REST.

GABARITO

1. Errado
2. Letra D
3. Letra D
4. Letra B
5. Letra A
6. Certo
7. Letra B
8. Letra B
9. Letra A
10. Letra E
11. Letra E
12. Letra A
13. Letra A
14. Letra D
15. Letra C
16. Letra D

ESSA LEI TODO MUNDO CONHECE:
PIRATARIA É CRIME.

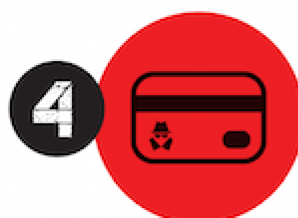
Mas é sempre bom revisar o porquê e como você pode ser prejudicado com essa prática.



Professor investe seu tempo para elaborar os cursos e o site os coloca à venda.

Pirata divulga ilicitamente (grupos de rateio), utilizando-se do anonimato, nomes falsos ou laranjas (geralmente o pirata se anuncia como formador de "grupos solidários" de rateio que não visam lucro).

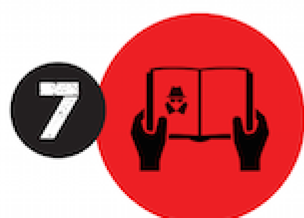
Pirata cria alunos fake praticando falsidade ideológica, comprando cursos do site em nome de pessoas aleatórias (usando nome, CPF, endereço e telefone de terceiros sem autorização).



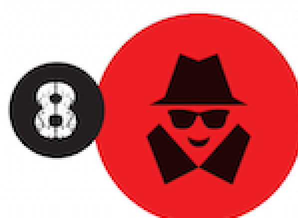
Pirata compra, muitas vezes, clonando cartões de crédito (por vezes o sistema anti-fraude não consegue identificar o golpe a tempo).

Pirata fere os Termos de Uso, adultera as aulas e retira a identificação dos arquivos PDF (justamente porque a atividade é ilegal e ele não quer que seus fakes sejam identificados).

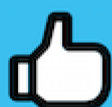
Pirata revende as aulas protegidas por direitos autorais, praticando concorrência desleal e em flagrante desrespeito à Lei de Direitos Autorais (Lei 9.610/98).



Concurseiro(a) desinformado participa de rateio, achando que nada disso está acontecendo e esperando se tornar servidor público para exigir o cumprimento das leis.



O professor que elaborou o curso não ganha nada, o site não recebe nada, e a pessoa que praticou todos os ilícitos anteriores (pirata) fica com o lucro.



Deixando de lado esse mar de sujeira, aproveitamos para agradecer a todos que adquirem os cursos honestamente e permitem que o site continue existindo.