

Em qualquer requisição HTTP válida, alguns componentes são obrigatórios e formam a estrutura mínima da requisição. Assinale **todos os componentes que estão presentes em toda requisição HTTP**:

Escolha uma ou mais:

- O caminho (URL) para o recurso que está sendo solicitado.
- Parâmetros de requisição, como ?pagina=1, são obrigatórios em todas as requisições HTTP.
- O método HTTP (como GET, POST, etc.), que define qual ação será executada no recurso.
- O corpo da requisição (body), onde os dados são enviados.
- Cabeçalhos HTTP mínimos, como o cabeçalho Host, que identifica o servidor de destino.

Repositórios definidos como interfaces que estendem JpaRepository são, por padrão, gerenciados como beans singleton pelo Spring.

- Verdadeiro
- Falso

Considere a entidade `Livro` com os seguintes atributos:

```
public class Livro {  
    private Long id;  
    private String titulo;  
    private String autor;  
    private Integer anoPublicacao;  
}
```

Quais dos métodos abaixo seriam válidos para declarar em uma interface `LivroRepository`, utilizando **Dynamic Finders** do Spring Data JPA?

Escolha uma ou mais:

- findByTituloContainingIgnoreCase
- findByAutorOrderByAnoPublicacaoAsc
- findLivroTitulo
- findByAnoPublicacaoBetween
- buscarPorAutorOrdenadoPorAno

Sobre o uso de anotações na definição de entidades em projetos com Spring Data JPA, assinale **todas as alternativas corretas**:

Escolha uma ou mais:

- A anotação `@RestController` é usada nas entidades para que elas possam ser expostas diretamente via API.
- A anotação `@Id` define qual atributo será utilizado como chave primária da entidade.
- A anotação `@Autowired` deve ser usada nos atributos da entidade para injetar os repositórios necessários.
- A anotação `@GeneratedValue` permite que a chave primária seja gerada automaticamente pelo banco ou provedor JPA.
- A anotação `@Entity` é usada para indicar que uma classe representa uma tabela no banco de dados.

Analise as afirmações a seguir sobre boas práticas e responsabilidades dos **Controllers** em uma API REST com Spring Boot. Assinale **todas as alternativas corretas**:

Escolha uma ou mais:

- O Controller atua como um intermediador: recebe requisições, valida dados e repassa para o Service tratar a lógica de negócio.
- Controllers devem receber e validar os dados de entrada, utilizando anotações como @Valid em conjunto com DTOs.
- É recomendável que os Controllers contenham diretamente a lógica de acesso ao banco para simplificar o código.
- Controllers devem sempre retornar diretamente entidades JPA para evitar mapeamentos e simplificar o código.
- Controllers devem retornar respostas apropriadas à API, podendo usar objetos como ResponseEntity para controlar status, cabeçalhos e corpo.

Em APIs REST, o design das URLs (endpoints) deve seguir algumas boas práticas para manter a API semântica, padronizada e de fácil entendimento. Assinale **todas as alternativas corretas**:

Escolha uma ou mais:

- Evita-se o uso de verbos nas URLs, pois a ação é representada pelo método HTTP (GET, POST, etc).
- É recomendado usar URLs como /getLivros ou /criarUsuario para deixar claro o que a API faz.
- Hierarquia entre recursos pode ser representada nas URLs, como em /usuarios/42/livros.
- As URLs em uma API REST devem usar substantivos no plural para representar coleções de recursos, como /livros ou /usuarios.
- Deve-se incluir o tipo de retorno na URL, como /livros.json ou /usuarios.xml.

Considere as afirmações abaixo sobre a camada **Repository** em uma aplicação Spring Boot utilizando Spring Data JPA. Assinale **todas as alternativas corretas**:

Escolha uma ou mais:

- Em Spring Boot, Repositories são definidos como interfaces que estendem JpaRepository ou outras interfaces do Spring Data.
- Repositories devem conter lógica de negócio e regras específicas da aplicação.
- É possível criar métodos de consulta personalizados apenas seguindo convenções de nomes, como findByEmail.
- É necessário implementar manualmente todos os métodos de CRUD em uma interface Repository.
- Repositories são normalmente injetados nas Services para realizar operações de persistência.

O padrão Strategy permite que um objeto altere seu comportamento em tempo de execução, ao trocar dinamicamente a estratégia utilizada.

Verdadeiro

Falso

Sobre funcionalidades avançadas da camada **Repository** com Spring Data JPA, assinale **todas as alternativas corretas**:

Escolha uma ou mais:

- Todos os métodos customizados devem ser implementados manualmente dentro de uma classe concreta, não sendo possível usar anotações.
- É possível executar queries SQL nativas em um Repository utilizando @Query com o atributo nativeQuery = true.
- Repositories não suportam paginação ou ordenação nativamente.
- É possível retornar apenas partes da entidade ou projeções customizadas usando interfaces ou DTOs nos métodos do Repository.
- A anotação @Query pode ser usada para definir consultas JPQL diretamente em métodos do Repository.

A camada Service pode conter lógica de apresentação, como formatação de HTML ou manipulação direta de objetos HTTP.

Verdadeiro

Falso

Um Controller no Spring Boot deve receber as requisições HTTP, validar os dados e delegar a lógica de negócio para a camada Service.

Verdadeiro

Falso

Considere a resposta abaixo, retornada por uma API REST após uma requisição:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 58
```

```
{
  "id": 101,
  "nome": "João",
  "email": "joao@email.com"
}
```

Com base nessa resposta, assinale **todas as afirmações corretas**:

Escolha uma ou mais:

- O cabeçalho Content-Type informa que o corpo da resposta está no formato JSON.
- O cabeçalho Content-Length indica a quantidade de campos retornados no JSON.
- O corpo da resposta contém os dados do recurso solicitado, no caso, um usuário com nome, e-mail e ID.
- O código 200 OK indica que a requisição foi processada com sucesso.
- Como o código 200 foi retornado, o corpo obrigatoriamente deve conter um token JWT para autenticação.

Spring Data JPA permite retornar apenas campos específicos da entidade usando interfaces como projeções, sem a necessidade de escrever consultas manuais.

Verdadeiro

Falso

O padrão de projeto **Observer** é usado para criar um sistema de notificação automática entre objetos. Assinale **todas as alternativas corretas** sobre esse padrão:

Escolha uma ou mais:

- O padrão depende de verificação constante (polling) pelos observadores para detectar mudanças no estado.
- O padrão Observer permite que um objeto notifique automaticamente uma lista de outros objetos quando seu estado muda.
- Todos os observadores devem executar exatamente a mesma ação ao serem notificados.
- O padrão Observer é considerado um padrão estrutural, pois define a organização das classes.
- Esse padrão reduz o acoplamento entre quem gera um evento e quem responde a ele.

É considerado boa prática que um Controller acesse diretamente o banco de dados usando Repositories.

Verdadeiro

Falso

Considere a seguinte resposta HTTP, retornada por uma API ao tentar cadastrar um e-mail que já existe:

```
HTTP/1.1 409 Conflict
Content-Type: application/json
Content-Length: 88

{
  "erro": "E-mail já cadastrado.",
  "codigo": "EMAIL_DUPLICADO",
  "timestamp": "2025-04-05T10:30:00Z"
}
```

Com base nessa resposta, assinale **todas as afirmações corretas**:

Escolha uma ou mais:

- O código 409 indica que o recurso solicitado não foi encontrado.
- O cabeçalho Content-Length define o número de linhas retornadas no corpo da resposta.
- O corpo da resposta está estruturado em JSON e fornece detalhes sobre o erro ocorrido.
- É uma boa prática que APIs retornem mensagens de erro claras e estruturadas, como visto nessa resposta.
- O código 409 Conflict indica que houve um conflito de dados ao processar a requisição.

Sobre o uso e papel da camada **Controller** em uma aplicação Spring Boot, assinale **todas as alternativas corretas**:

Escolha uma ou mais:

- Controllers devem conter toda a lógica de negócio, pois estão mais próximos do cliente.
- O Controller atua como ponto de entrada da aplicação web, lidando com verbos HTTP como GET, POST, PUT e DELETE.
- O Controller deve acessar diretamente o banco de dados para retornar as informações solicitadas.
- O Controller é responsável por lidar com requisições HTTP e delegar a lógica de negócio para a camada Service.
- Controllers em APIs REST geralmente são anotados com @RestController, que combina @Controller e @ResponseBody.

Ao implementar autenticação em APIs REST, algumas práticas e códigos HTTP são utilizados de forma padronizada. Assinale **todas as alternativas corretas**:

Escolha uma ou mais:

- JWTs (JSON Web Tokens) são considerados tokens translúcidos, pois podem ser lidos e decodificados sem contato com o servidor.
- Tokens opacos são preferíveis quando o cliente precisa acessar diretamente os dados do token, como permissões e ID do usuário.
- Ao realizar login com sucesso, a API pode retornar um código 200 OK e incluir um token no corpo da resposta.
- O código 403 Forbidden deve ser usado quando o login falha por usuário ou senha incorretos.
- O código 401 Unauthorized deve ser usado quando o usuário não fornece ou fornece um token de autenticação inválido.

Considere as seguintes afirmações sobre a **camada Service em uma aplicação Spring Boot**. Assinale **todas as alternativas corretas**:

Escolha uma ou mais:

- a. É uma boa prática anotar as classes da camada Service com @Service para que o Spring as registre como beans gerenciados.
- b. Uma classe Service pode depender de outras classes Service para organizar melhor responsabilidades.
- c. A camada Service deve conter regras de negócio e orquestrar chamadas para outras camadas, como repositórios e APIs externas.
- d. A camada Service deve conter apenas chamadas de banco de dados e não pode conter nenhuma lógica de negócio.
- e. A camada Service é responsável por receber requisições HTTP diretamente dos clientes e encaminhá-las para o Repository.

Swagger é uma ferramenta amplamente utilizada para documentar e testar APIs REST. Com base nisso, assinale **todas as alternativas corretas**:

Escolha uma ou mais:

- Swagger é obrigatório em qualquer projeto REST que utiliza Spring Boot.
- Em projetos Spring Boot, é possível integrar Swagger utilizando bibliotecas como springdoc-openapi.
- Swagger só funciona se todas as classes da API estiverem anotadas com @RestController.
- Swagger/OpenAPI permite gerar uma documentação interativa que descreve os endpoints da API, métodos HTTP, parâmetros e respostas.
- Swagger substitui totalmente a necessidade de escrever testes automatizados na aplicação.

Associe cada descrição à ação HTTP (verbo) mais adequada para uma API REST.

Verificar quais métodos HTTP são permitidos em um determinado endpoint

OPTIONS ▾

Solicitar os dados de um recurso, sem alterar nada no servidor

GET ▾

Atualizar completamente um recurso existente, substituindo todos os dados

PUT ▾

Atualizar parcialmente um recurso existente, modificando apenas alguns campos

PATCH ▾

Criar um novo recurso no servidor

POST ▾

Remover um recurso existente do servidor

DELETE ▾

Sobre o uso de **Service** no Spring Boot e as anotações do framework, assinale **todas as alternativas corretas**:

Escolha uma ou mais:

- Classes anotadas com @Service são, por padrão, beans singleton no Spring.
- A injeção de dependência em Services pode ser feita via construtor, sendo essa a abordagem mais recomendada.
- A anotação @Service registra a classe no contexto do Spring para ser gerenciada como um bean.
- O Spring exige que as Services sejam instanciadas manualmente usando a palavra-chave new.
- Para que o Spring reconheça uma classe como Service, é obrigatório criar um arquivo XML de configuração manual.

Associe cada descrição abaixo ao código HTTP correto utilizado em APIs REST.

Requisição inválida, dados malformados ou parâmetros ausentes

400



Recurso criado com sucesso, normalmente após uma operação POST

201



Recurso solicitado não foi encontrado no servidor

404



Requisição bem-sucedida com retorno de conteúdo no corpo da resposta

200



Falha de autenticação, como token ausente ou inválido

401



Requisição processada com sucesso, mas sem corpo na resposta

204



Sobre o uso e papel da camada **Controller** em uma aplicação Spring Boot, assinale **todas as alternativas corretas**:

Escolha uma ou mais:

- Controllers em APIs REST geralmente são anotados com @RestController, que combina @Controller e @ResponseBody.
- O Controller é responsável por lidar com requisições HTTP e delegar a lógica de negócios para a camada Service.
- O Controller deve acessar diretamente o banco de dados para retornar as informações solicitadas.
- O Controller atua como ponto de entrada da aplicação web, lidando com verbos HTTP como GET, POST, PUT e DELETE.
- Controllers devem conter toda a lógica de negócios, pois estão mais próximos do cliente.

Associe cada descrição abaixo à anotação de validação (Bean Validation) correspondente.

O campo não pode ser vazio e deve conter caracteres visíveis	@NotBlank
O campo deve conter um e-mail válido	@Email
O campo deve ter tamanho dentro de um intervalo de caracteres	@Size
O número deve ser positivo (maior que zero)	@Positive
O campo deve ser nulo (usado em campos controlados pelo sistema)	@Null
O número decimal deve ser maior ou igual a um valor mínimo	@DecimalMin
O valor deve obedecer a um formato definido por expressão regular	@Pattern
O número inteiro deve ser menor ou igual a um valor	@Max
O número pode ser zero ou negativo	@NegativeOrZero
O número inteiro deve ser maior ou igual a um valor	@Min
O valor deve ser uma data no passado	@Past
O valor deve ser uma data no futuro	@Future
O campo não pode ser nulo	@NotNull
O número pode ser zero ou positivo	@PositiveOrZero

É permitido e, em muitos casos, recomendado que uma classe Service chame outra classe Service quando há lógica de negócio compartilhada.

Verdadeiro

Falso

Um Controller pode receber instâncias de Services via injeção de dependência, seja por construtor ou com a anotação @Autowired.

Verdadeiro

Falso

Considere as afirmações abaixo sobre a camada **Repository** em uma aplicação Spring Boot utilizando Spring Data JPA. Assinale **todas as alternativas corretas**:

Escolha uma ou mais:

- É possível criar métodos de consulta personalizados apenas seguindo convenções de nomes, como findByEmail.
- É necessário implementar manualmente todos os métodos de CRUD em uma interface Repository.
- Repositories devem conter lógica de negócio e regras específicas da aplicação.
- Repositories são normalmente injetados nas Services para realizar operações de persistência.
- Em Spring Boot, Repositories são definidos como interfaces que estendem JpaRepository ou outras interfaces do Spring Data.

Sobre o conceito de injeção de dependência no Spring Framework e o uso da anotação **@Autowired**, assinale **todas as alternativas corretas**:

Escolha uma ou mais:

- Para usar @Autowired corretamente, é necessário instanciar os objetos manualmente com new.
- @Autowired funciona normalmente em qualquer classe Java, mesmo fora de um projeto Spring.
- O uso de @Autowired exige que a classe anotada esteja definida como final.
- A anotação @Autowired pode ser utilizada para injetar dependências em atributos, métodos ou construtores.
- O uso de injeção de dependência ajuda a reduzir o acoplamento entre as classes, facilitando testes e manutenção.

Considere a entidade `Livro` com os seguintes atributos:

```
public class Livro {  
    private Long id;  
    private String titulo;  
    private String autor;  
    private Integer anoPublicacao;  
}
```

Quais dos métodos abaixo seriam válidos para declarar em uma interface `LivroRepository`, utilizando **Dynamic Finders** do Spring Data JPA?

Escolha uma ou mais:

- buscarPorAutorOrdenadoPorAno
- findLivroTitulo
- findByAnoPublicacaoBetween
- findByAutorOrderByAnoPublicacaoAsc
- findByTituloContainingIgnoreCase

Em qualquer requisição HTTP válida, alguns componentes são obrigatórios e formam a estrutura mínima da requisição. Assinale **todos os componentes que estão presentes em toda requisição HTTP**:

Escolha uma ou mais:

- Cabeçalhos HTTP mínimos, como o cabeçalho Host, que identifica o servidor de destino.
- O corpo da requisição (body), onde os dados são enviados.
- Parâmetros de requisição, como ?pagina=1, são obrigatórios em todas as requisições HTTP.
- O caminho (URL) para o recurso que está sendo solicitado.
- O método HTTP (como GET, POST, etc.), que define qual ação será executada no recurso.

Repositórios definidos como interfaces que estendem JpaRepository são, por padrão, gerenciados como beans singleton pelo Spring.

Verdadeiro

Falso

Sobre o uso de anotações no Spring Framework, assinale **todas as alternativas corretas**:

Escolha uma ou mais:

- A anotação @Autowired pode ser usada normalmente em qualquer classe Java, mesmo fora de aplicações Spring.
- A anotação @Autowired é usada para injeção automática de dependências entre beans do Spring.
- A anotação @Service permite que uma classe seja detectada automaticamente como bean pelo Spring.
- Toda classe em um projeto Spring precisa ser anotada com @Component para funcionar.
- A anotação @Repository é recomendada para classes que realizam operações com o banco de dados.

Considere as anotações fornecidas pelo Spring Boot comumente utilizadas em Controllers REST. Assinale **todas as alternativas corretas**:

Escolha uma ou mais:

- @GetMapping é uma especialização de @RequestMapping usada para mapear requisições HTTP GET.
- @PathVariable serve para injetar configurações do application.properties no Controller.
- @Entity é uma anotação usada em métodos de Controller para transformar respostas em entidades do banco.
- @RequestMapping pode ser usada para mapear classes ou métodos a caminhos e verbos HTTP.
- @RequestBody é usada para vincular o corpo da requisição a um parâmetro de método em um Controller.

Sobre boas práticas e responsabilidades da **camada Service em uma aplicação Spring Boot**, assinale **todas as alternativas corretas**:

Escolha uma ou mais:

- A Service não deve acessar diretamente objetos da camada de visualização (como DTOs ou objetos HTTP), mantendo a separação de responsabilidades.
- É recomendável que a Service manipule diretamente os dados da requisição (DTO de entrada) e da resposta (DTO de saída).
- A Service deve conter anotações como @Entity e @Id para persistência correta no banco de dados.
- É comum que uma Service reutilize lógica de negócio agrupando funcionalidades que podem ser compartilhadas entre diferentes Controllers.
- A injeção de dependência em Services deve ser feita preferencialmente via construtor, pois isso facilita testes e manutenção.

Sobre a integração dos Controllers com o Spring Framework

- Controllers devem ser criados como classes estáticas para permitir que o Spring as injete automaticamente. ✗
 - Controllers podem delegar chamadas para Repositories indiretamente, utilizando Services como intermediários. ✓
 - É possível injetar Services diretamente em Controllers usando a anotação @Autowired ou injeção via construtor. ✓
 - Controllers precisam ser registrados manualmente no Spring via arquivos XML de configuração. ✗
 - Controllers são beans gerenciados pelo Spring, detectados automaticamente via anotações como @RestController. ✓
-

Sobre a camada Service em uma aplicação Spring Boot

- A camada Service deve conter apenas chamadas de banco de dados e não pode conter nenhuma lógica de negócio. ✗
 - A camada Service deve conter regras de negócio e orquestrar chamadas para outras camadas, como repositórios e APIs externas. ✓
 - Uma classe Service pode depender de outras classes Service para organizar melhor responsabilidades. ✓
 - É uma boa prática anotar as classes da camada Service com @Service para que o Spring as registre como beans gerenciados. ✓
 - A camada Service é responsável por receber requisições HTTP diretamente dos clientes e encaminhá-las para o Repository. ✗
-

Sobre cabeçalhos HTTP em APIs REST

- O cabeçalho Host define quais campos devem ser obrigatórios em um formulário enviado via POST. ✗
- O cabeçalho Authorization é usado para enviar tokens de autenticação, como Bearer tokens em APIs REST. ✓
- O cabeçalho Content-Type indica o formato dos dados enviados no corpo da requisição, como application/json. ✓

- O cabeçalho Accept é usado pelo cliente para indicar qual formato deseja receber na resposta, como JSON ou XML. ✓
 - Cabeçalhos HTTP são ignorados pelas APIs REST, já que toda a informação necessária está na URL ou no corpo da requisição. ✗
-

Sobre ORM (Mapeamento Objeto-Relacional)

- O uso de ORM elimina automaticamente qualquer problema de performance causado por consultas ao banco. ✗
- O uso de ORM automatiza operações como inserção, atualização, deleção e consulta de registros no banco de dados. ✓
- Com ORM, não é mais necessário aprender ou compreender a linguagem SQL. ✗
- ORM ignora o modelo relacional, pois trabalha exclusivamente com objetos. ✗
- A técnica ORM permite mapear classes e atributos de uma linguagem orientada a objetos para tabelas e colunas de um banco de dados relacional. ✓

É considerado boa prática que um Controller acesse diretamente o banco de dados usando Repositories.

Verdadeiro

Falso

Os códigos da família **4xx** indicam que houve um erro por parte do cliente em uma requisição HTTP. Assinale **todas as alternativas corretas** sobre o uso desses códigos em APIs REST:

Escolha uma ou mais:

- O código 422 Unprocessable Entity deve ser usado para indicar que o servidor teve um erro interno ao processar a requisição.
- O código 404 Not Found é utilizado quando o recurso solicitado não existe na API.
- O código 401 Unauthorized é usado quando o usuário está autenticado, mas não tem permissão para acessar o recurso.
- O código 400 Bad Request é usado quando a requisição está malformada ou os dados enviados são inválidos.
- O código 409 Conflict é apropriado quando há um conflito de estado, como tentar cadastrar um e-mail já existente.

Associe cada descrição ao padrão de projeto correspondente.

Permite integrar duas interfaces incompatíveis sem alterar o código original

Adapter

Permite trocar dinamicamente o algoritmo usado em um objeto

Strategy

Permite que múltiplos objetos sejam notificados automaticamente quando outro objeto muda de estado

Observer

Um Controller no Spring Boot deve receber as requisições HTTP, validar os dados e delegar a lógica de negócio para a camada Service.

Verdadeiro

Falso

REST é um estilo arquitetural utilizado no desenvolvimento de APIs que se baseia em regras e restrições para promover simplicidade, escalabilidade e padronização. Assinale **todas as alternativas corretas** relacionadas a REST e sua utilização do protocolo HTTP:

Escolha uma ou mais:

- Em REST, cada recurso deve ser identificado por uma URL única que o represente.
- Uma API RESTful deve obrigatoriamente retornar todos os dados da aplicação em uma única chamada.
- APIs REST devem utilizar os códigos de status HTTP para informar o resultado das requisições, como 200, 201, 404, entre outros.
- REST é um padrão obrigatório definido por uma organização e deve seguir uma implementação oficial.
- Em uma API REST, as operações são representadas por métodos HTTP como GET, POST, PUT e DELETE.

Ao receber uma resposta de uma requisição HTTP, a estrutura da resposta segue um padrão definido pelo protocolo. Assinale **todos os componentes que estão presentes em toda resposta HTTP válida**:

Escolha uma ou mais:

- Cabeçalhos de resposta (headers), como Content-Type ou Content-Length.
- O corpo da resposta (body), com os dados retornados pela API, é sempre obrigatório.
- O código de status HTTP, como 200, 404 ou 500, indicando o resultado da requisição.
- Cookies devem ser incluídos em toda resposta HTTP para manter a sessão ativa.
- A versão do protocolo HTTP, como HTTP/1.1 ou HTTP/2.

Sobre boas práticas e responsabilidades da **camada Service em uma aplicação Spring Boot**, assinale **todas as alternativas corretas**:

Escolha uma ou mais:

- A injeção de dependência em Services deve ser feita preferencialmente via construtor, pois isso facilita testes e manutenção.
- A Service deve conter anotações como @Entity e @Id para persistência correta no banco de dados.
- É comum que uma Service reutilize lógica de negócio agrupando funcionalidades que podem ser compartilhadas entre diferentes Controllers.
- É recomendável que a Service manipule diretamente os dados da requisição (DTO de entrada) e da resposta (DTO de saída).
- A Service não deve acessar diretamente objetos da camada de visualização (como DTOs ou objetos HTTP), mantendo a separação de responsabilidades.

O padrão de projeto **Strategy** permite encapsular algoritmos ou comportamentos que podem ser escolhidos em tempo de execução. Assinale **todas as alternativas corretas** sobre esse padrão:

Escolha uma ou mais:

- O padrão Strategy é usado exclusivamente em algoritmos de ordenação e busca.
- O padrão Strategy é apenas uma forma de organizar estruturas condicionais, como if/else.
- Strategy é útil quando temos diversas variações de um algoritmo que podem ser alternadas conforme o contexto.
- O padrão Strategy depende de uma hierarquia rígida de herança entre classes concretas.
- O padrão Strategy permite encapsular algoritmos diferentes e tornar o comportamento de um objeto intercambiável em tempo de execução.

Sobre o uso de **Service** no Spring Boot e as anotações do framework, assinale **todas as alternativas corretas:**

Escolha uma ou mais:

- Para que o Spring reconheça uma classe como Service, é obrigatório criar um arquivo XML de configuração manual.
- A anotação @Service registra a classe no contexto do Spring para ser gerenciada como um bean.
- Classes anotadas com @Service são, por padrão, beans singleton no Spring.
- A injeção de dependência em Services pode ser feita via construtor, sendo essa a abordagem mais recomendada.
- O Spring exige que as Services sejam instanciadas manualmente usando a palavra-chave new.

Considere a seguinte resposta HTTP, retornada por uma API ao tentar cadastrar um e-mail que já existe:

```
HTTP/1.1 409 Conflict
Content-Type: application/json
Content-Length: 88
```

```
{
  "erro": "E-mail já cadastrado.",
  "codigo": "EMAIL_DUPLICADO",
  "timestamp": "2025-04-05T10:30:00Z"
}
```

Com base nessa resposta, assinale **todas as afirmações corretas:**

Escolha uma ou mais:

- O código 409 Conflict indica que houve um conflito de dados ao processar a requisição.
- O cabeçalho Content-Length define o número de linhas retornadas no corpo da resposta.
- É uma boa prática que APIs retornem mensagens de erro claras e estruturadas, como visto nessa resposta.
- O corpo da resposta está estruturado em JSON e fornece detalhes sobre o erro ocorrido.
- O código 409 indica que o recurso solicitado não foi encontrado.

A camada Service pode conter lógica de apresentação, como formatação de HTML ou manipulação direta de objetos HTTP.

- Verdadeiro
 Falso

Sobre a integração dos Controllers com o Spring Framework, assinale **todas as alternativas corretas:**

Escolha uma ou mais:

- É possível injetar Services diretamente em Controllers usando a anotação @Autowired ou injeção via construtor.
- Controllers podem delegar chamadas para Repositories indiretamente, utilizando Services como intermediários.
- Controllers são beans gerenciados pelo Spring, detectados automaticamente via anotações como @RestController.
- Controllers devem ser criados como classes estáticas para permitir que o Spring as injete automaticamente.
- Controllers precisam ser registrados manualmente no Spring via arquivos XML de configuração.

Analise as afirmações a seguir sobre boas práticas e responsabilidades dos **Controllers** em uma API REST com Spring Boot. Assinale **todas as alternativas corretas**:

Escolha uma ou mais:

- Controllers devem sempre retornar diretamente entidades JPA para evitar mapeamentos e simplificar o código.
- O Controller atua como um intermediador: recebe requisições, valida dados e repassa para o Service tratar a lógica de negócio.
- É recomendável que os Controllers contenham diretamente a lógica de acesso ao banco para simplificar o código.
- Controllers devem retornar respostas apropriadas à API, podendo usar objetos como ResponseEntity para controlar status, cabeçalhos e corpo.
- Controllers devem receber e validar os dados de entrada, utilizando anotações como @Valid em conjunto com DTOs.

Sobre DTOs, é correto afirmar:

- I. É o anagrama para Data Transfer Object. São classes criadas para não terem nenhuma regra complexa nelas. Basicamente possuem atributos e getters/setters.
- II. É um padrão de projetos que só existe em Java e só deve ser aplicado em projetos REST.
- III. Podemos usar anotações de validação em DTO's, como @NotBlank, @Min, @Past etc.
- IV. É apenas outro nome para as classes de Entidade, usadas para mapear as tabelas em classes.
- V. Serve para transferir dados, entre camadas da própria aplicação e/ou para outras aplicações.

- a. Apenas II, IV e V estão corretas.
- b. Apenas II, III e IV estão corretas.
- c. Apenas I, II e V estão corretas.
- d. Apenas I, III e V estão corretas.
- e. Nenhuma das alternativas.

[Limpar minha escolha](#)