Certified Tech Developer
The Ultimate Degree

Data Transfer Object (DTO)

Um dos problemas mais comuns quando desenvolvemos aplicações é o design de como as informações devem passar de uma camada de serviço (ou um controller) para as aplicações ou camada de apresentação. Muitas vezes, por falta de conhecimento, utilizamos as classes de entidades para retornar os dados. Isso ocasiona no envio de mais dados do que o necessário ou até mesmo ter que acessar a camada de serviço mais de uma vez para recuperar os dados necessários.

Para facilitar este trabalho, existe o padrão **DTO**.

O objetivo do padrão **DTO** é criar um objeto plano (**POJO**) com uma série de atributos que podem ser enviados e recuperados do servidor em uma única chamada, de forma que o **DTO** possa conter informações de várias fontes ou tabelas agrupando em uma única classe simples.

Embora um DTO seja apenas um objeto plano, ele deve seguir algumas regras:

Entidades versus DTO

Um erro muito frequente é o fato de utilizar as classes de entidades para a transmissão de dados entre o cliente e o servidor. No entanto, entidades são classes que foram projetadas para mapear o banco de dados, não para serem uma visualização para a tela ou serviço específico, o que faz com que muitos dos campos não sejam serializáveis ou não contenham todos os campos necessários.

O fato de as entidades não conterem todos os atributos necessários, ou de não serem serializáveis, traz outros problemas. Por exemplo, a necessidade de adicionar mais atributos às entidades com o único objetivo de poder cobrir os requisitos de transferência de dados, deixando

de lado o verdadeiro propósito da entidade, que é apenas mapear o banco de dados, o que induz a criação de uma mistura entre entidade e DTO.

ResponseEntity

Com o Spring, geralmente temos muitas formas de atingir o mesmo objetivo, incluindo o ajuste fino de respostas HTTP.

Vejamos a classe **ResponseEntity< T >.**

Alternativas

- @ResponseBody: em aplicações clássicas com o Spring MVC, os endpoints geralmente retornam páginas HTML renderizadas, as vezes, só precisamos retornar os dados.

  Nesses casos, podemos marcar o método do controller com @ResponseBody, e o Spring trata o valor do resultado do métodos como o próprio corpo da resposta HTTP.

- @ResponseStatus: para quando um endpoint retorna com sucesso, o Spring fornece uma resposta HTTP 200 (OK). Se o endpoint lançar uma exceção, o Spring procura por um manipulador de exceção que indica qual status HTTP usar. Podemos marcar o método com @ResponseStatus e, portanto, o Spring retorna com um status HTTP personalizado.

- Manipular diretamente a resposta: o Spring também nos permite acessar diretamente o objeto javax.servlet.http.HttpServletResponse, apenas temos que declará-lo como um argumento do método, como vemos no código abaixo:

```
@GetMapping("/manual")
void manual (HttpServerletResponse response) throws IOException
{
        response.setHeader("Custom - Header", "foo");
        response.setStatus(200);
        response.getWriter().println("Hello Word!");

}
```