Estamos utilizando DTOs para representar os dados que recebemos e devolvemos pela API, mas você provavelmente deve estar se perguntando “Por que ao invés de criar um DTO não devolvemos diretamente a entidade JPA no Controller?”. Para fazer isso, bastaria alterar o método listar no Controller para:

@GetMapping

**public** List<Medico> **listar**() {

**return** repository.findAll();

}COPIAR CÓDIGO

Desse jeito o código ficaria mais enxuto e não precisaríamos criar o DTO no projeto. Mas, será que isso realmente é uma boa ideia?

**Os problemas de receber/devolver entidades JPA**

De fato é muito mais simples e cômodo não utilizar DTOs e sim lidar diretamente com as entidades JPA nos controllers. Porém, essa abordagem tem algumas desvantagens, inclusive causando vulnerabilidade na aplicação para ataques do tipo [**Mass Assignment**](https://cheatsheetseries.owasp.org/cheatsheets/Mass_Assignment_Cheat_Sheet.html).

Um dos problemas consiste no fato de que, ao retornar uma entidade JPA em um método de um Controller, o Spring vai gerar o JSON contendo **todos** os atributos dela, sendo que nem sempre esse é o comportamento que desejamos.

Eventualmente podemos ter atributos que não desejamos que sejam devolvidos no JSON, seja por motivos de segurança, no caso de dados *sensíveis*, ou mesmo por não serem utilizados pelos clientes da API.

**Utilização da anotação @JsonIgnore**

Nessa situação, poderíamos utilizar a anotação @JsonIgnore, que nos ajuda a ignorar certas propriedades de uma classe Java quando ela for serializada para um objeto JSON.

Sua utilização consiste em adicionar a anotação nos atributos que desejamos ignorar quando o JSON for gerado. Por exemplo, suponha que em um projeto exista uma entidade JPA Funcionario, na qual desejamos ignorar o atributo salario:

@Getter

@NoArgsConstructor

@EqualsAndHashCode(of = "id")

@Entity(name = "Funcionario")

@Table(name = "funcionarios")

public class Funcionario {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

**private** **String** **nome**;

**private** **String** **email**;

@**JsonIgnore**

**private** **BigDecimal** **salario**;

//restante do código omitido…

}COPIAR CÓDIGO

No exemplo anterior, o atributo salario da classe Funcionario não será exibido nas respostas JSON e o problema estaria solucionado.

Entretanto, pode acontecer de existir algum outro endpoint da API na qual precisamos enviar no JSON o salário dos funcionários, sendo que nesse caso teríamos problemas, pois com a anotação @JsonIgnore tal atributo **nunca** será enviado no JSON, e ao remover a anotação o atributo **sempre** será enviado. Perdemos, com isso, a flexibilidade de controlar quando determinados atributos devem ser enviados no JSON e quando não.

**DTO**

O padrão DTO (*Data Transfer Object*) é um padrão de arquitetura que era bastante utilizado antigamente em aplicações Java distribuídas (arquitetura cliente/servidor) para representar os dados que eram enviados e recebidos entre as aplicações cliente e servidor.

O padrão DTO pode (e deve) ser utilizado quando não queremos expor todos os atributos de alguma entidade do nosso projeto, situação igual a dos salários dos funcionários mostrado no exemplo de código anterior. Além disso, com a flexibilidade e a opção de filtrar quais dados serão transmitidos, podemos poupar tempo de processamento.

**Loop infinito causando StackOverflowError**

Outro problema muito recorrente ao se trabalhar diretamente com entidades JPA acontece quando uma entidade possui algum autorrelacionamento ou relacionamento bidirecional. Por exemplo, considere as seguintes entidades JPA:

@Getter

@NoArgsConstructor

@EqualsAndHashCode(of = "id")

@Entity(name = "Produto")

@Table(name = "produtos")

**public** **class** **Produto** {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

**private** Long id;

**private** String nome;

**private** String descricao;

**private** BigDecimal preco;

@ManyToOne

@JoinColumn(name = “id\_categoria”)

**private** Categoria categoria;

//restante do código omitido…

}COPIAR CÓDIGO

@Getter

@NoArgsConstructor

@EqualsAndHashCode(of = "id")

@Entity(name = "Categoria")

@Table(name = "categorias")

**public** **class** **Categoria** {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

**private** Long id;

**private** String nome;

@OneToMany(mappedBy = “categoria”)

**private** List<Produto> produtos = **new** **ArrayList**<>();

//restante do código omitido…

}COPIAR CÓDIGO

Ao retornar um objeto do tipo Produto no Controller, o Spring teria problemas para gerar o JSON desse objeto, causando uma exception do tipo StackOverflowError. Esse problema ocorre porque o objeto produto tem um atributo do tipo Categoria, que por sua vez tem um atributo do tipo List<Produto>, causando assim um loop infinito no processo de serialização para JSON.

Tal problema pode ser resolvido com a utilização da anotação @JsonIgnore ou com a utilização das anotações @JsonBackReference e @JsonManagedReference, mas também poderia ser evitado com a utilização de um DTO que representa apenas os dados que devem ser devolvidos no JSON.