JSON

Porque usar JSON ?

Imagine que você precise desenvolver a funcionalidade de gerenciar a tomada de livros por empréstimo em uma biblioteca para uma nova *fintech* com o objetivo de fazer com que o tráfego de dados fosse o mínimo possível para que tudo continuasse funcionando bem mesmo em redes mais lentas ou com largura de banda menor.

Utilizamos o formato JSON **como uma alternativa ao XML para a transferência de dados estruturados entre um servidor de Web e uma aplicação Web**.

O que é JSON ?

O formato **JSON** (*JavaScript Object Notation*) é, como o nome sugere, uma forma de notação de objetos **JavaScript** de modo que eles possam ser representados de uma forma comum a diversas linguagens.

Além disso, uma ideia que está fortemente enraizada neste formato é que ele seja facilmente trafegado entre aplicações em quaisquer protocolos, inclusive o **HTTP.** Portanto, a principal diferença entre um objeto JavaScript padrão e um JSON é o fato do JSON ser na realidade: um texto.

Como usá-lo ?

Um **JSON** deve conter apenas informações que possam ser representadas em formato de texto. Listei algumas regras:

- Não pode conter funções;
- Não pode conter comentários;
- Todo texto sempre tem aspas duplas;
- As propriedades sempre tem aspas duplas.

Desta forma, imagine o envio de uma tomada de um novo livro para um cliente, com um identificador numérico qualquer do cliente e uma lista de livros a serem feitos no pedido em questão. Tais informações teriam, em **JSON**, o seguinte formato:

```
{
    "id": 1,
    "nome": "Fabio"
    "livros": [
                {
                    "id": 1,
                    "nome": "1984"
                },
                {
                     "id": 2,
                    "nome": "Fazendo dos animais"
                }
     ],

}
```

O JSON gerado pelo framework segue o formato da classe modelo, contendo todos os atributos que possuam métodos públicos getters e setters e que não possuírem a anotação @JsonIgnore. O JSON anterior é gerado a partir do modelo da Entidade Cliente e Livro.

```java
@Entity
@Table(name = "cliente")
public class Cliente {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotNull
    @Column(name = "nome", length = 100)
    private String nome;
    @OneToMany(mappedBy = "cliente")
    private List<Livro> livros = new ArrayList<>();
    // Getters and Setters
}
```

```java
@Entity
@Table(name = "livro")
public class Livro {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Long id;

	@NotNull
	@Column(name = "nome", length = 100)
	private String nome;

	@JsonIgnore
	@ManyToOne
	@JoinColumn(name = "cliente_id")
	private Cliente cliente;
	// Getters and Setters
}
```