DTO, Response y POST

Response

En la clase anterior trabajamos sobre métodos GET que cuyas respuestas eran String o también conocido como **text/plain**

Hoy en día las API REST en su mayoría retornan contenido en formato de **JSON** aunque podemos encontrar algunos proyectos antiguos utilizando **XML.**

JSON es un **formato de intercambio de datos**, fácil de escribir y leer, que se utiliza hoy en día para compartir información entre diferentes aplicaciones y lenguajes. JSON permite pasar pares de valores, arrays y objetos, lo que le da una **gran capacidad de almacenar datos de todo tipo**, de forma fácil y comprensible, que lo ha hecho popular y ha sustituido al formato XML como estándar de intercambio.

DTO (Data Transfer Object):

Un DTO es **un objeto Java** utilizado para la transferencia de información, En Spring dentro del @RestController podemos retornar directamente un DTO y el **framework se encarga de transformarlo a formato JSON** y retornarlo. (application/json)

Vista del DTO en formato de JSON

```
[
        {
        "Date": date1,
        "Age": age1
        },
        { "Date": date2,
        "Age": age2
        },
        { "Date": date3,
        "Age": age3
        }
]
```

ResponseEntity

Representa la respuesta HTTP completa: código de estado, encabezados y cuerpo. Como resultado, podemos usarlo para configurar completamente la respuesta HTTP.

```java
@GetMapping("/hello")
ResponseEntity<String> hello() {
    return new ResponseEntity<>("Hello World!", HttpStatus.OK);
}
```

POST Method:

Utilizando **@PostMapping** generamos un endpoint sobre el método **HTTP POST**. Este nos permite poder ingresar parámetros utilizando el **HTTP BODY en formato JSON.**

Como podemos ver en el siguiente test donde modificamos el ejercicio anterior para que reciba **un DateDTO en formato JSON.**

@RequestBody lo utilizamos para informar que ese objeto lo vamos a asociar al HTTP BODY.

```
@Test
void shouldCalculateAgeFromDate() throws Exception {
  this.mockMvc.perform(post( urlTemplate: "/calculate")
      .contentType(MediaType.APPLICATION_JSON)
      .content("{\"day\":9,\"month\":9,\"year\":1989}"))
      .andDo(print())
      .andExpect(status().isOk())
      .andExpect(content().string(containsString( substring: "31")));
}
```

```
@PostMapping("/calculate")
public AgeDTO getAge(@RequestBody DateDTO dateDTO) {
```

RequestBody

Utilizando @RequestBody mapearemos el json y nos sirve para deserializar un objeto completo a partir del cuerpo de la petición. Si tenemos esta clase:

```
public class Person {
  private String name;
  private String surname;
  // Getters+setters, etc.
}
```

Y tenemos este método:

```
public String getData(@RequestBody Person person) { ... }
```

Spring en este caso nos inyectaría en la variable **person** un objeto de la clase **Person** con sus atributos que están informados con lo que no venga en el cuerpo de esta petición.

Payload en el uso de POST.

Como parte de una solicitud POST o PUT, se puede enviar una carga útil -o payload- de datos al servidor en el cuerpo de la solicitud.

El contenido del cuerpo puede ser cualquier objeto JSON válido, por ejemplo, como este:

```
{
   "FirstName": "Charly"
   "LastName" : "Arroyo",
   "UserName" : "charlyred",
   "Email"    :
"charlyred@digitalhouse.com"
 }
```

Métodos que usaremos en Spring

GET: utilizado para **consultar información** al servidor.

POST: Utilizado para la **creación de un nuevo registro.**

PUT: Se utiliza para **actualizar por completo** un registro existente.

PATCH: Similar al anterior, pero actualiza **solo un fragmento del registro.**

DELETE: Se utiliza para **eliminar un registro existente.**

HEAD: Se utiliza para o**btener información sobre un determinado recurso** sin retornar el registro.