

01-03-2021

Spring Platform

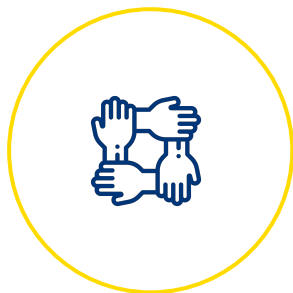
IT BOARDING

BOOTCAMP



WIP

// Índice



Multiples parametros



Respuestas



POST

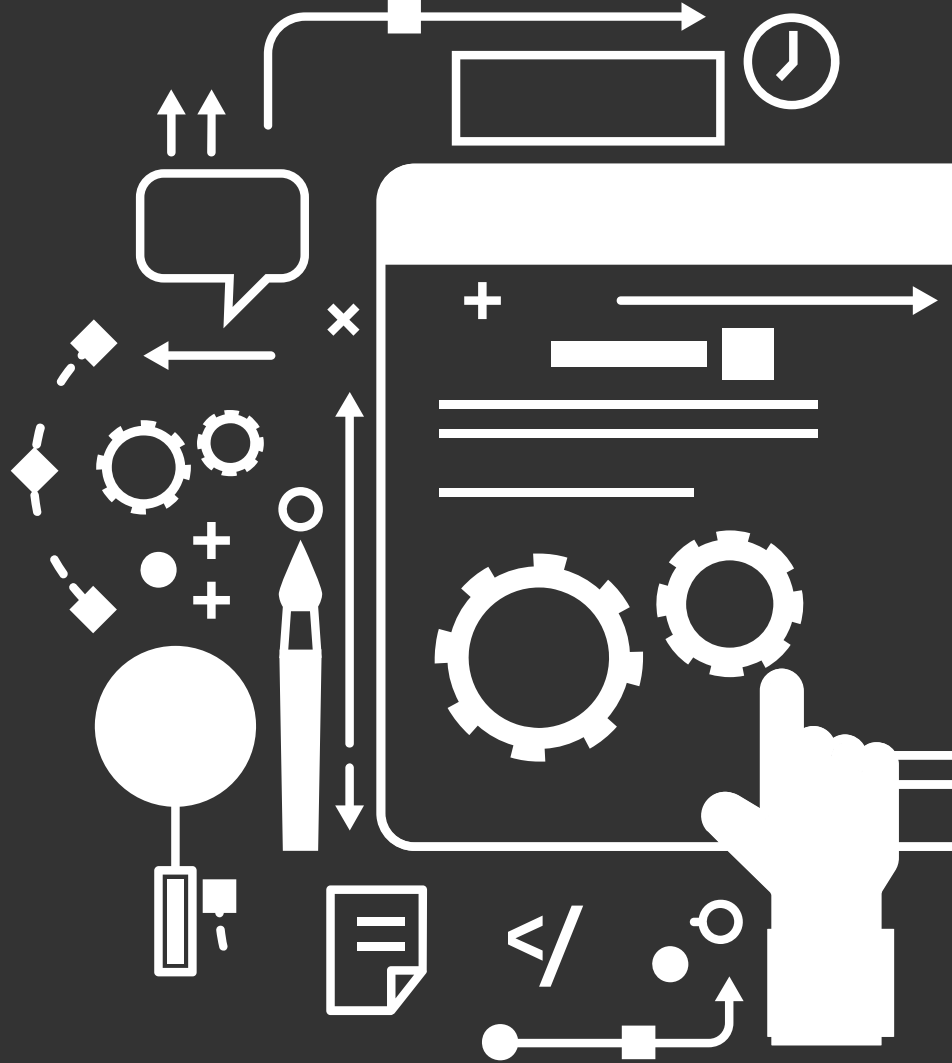
IT BOARDING

BOOTCAMP

Múltiples Parámetros

IT BOARDING

BOOTCAMP





Múltiples Parámetros

En un solo método de nuestra API podemos recibir tantos parámetros como lo necesitemos. Para demostrarlo vamos a crear una calculadora de edades que va a recibir una fecha de nacimiento y va a retornar la edad asociada

```
@GetMapping("/{day}/{month}/{year}")  
public String getAge(@PathVariable Integer day, @PathVariable Integer month, @PathVariable Integer year)
```

Como vemos en la imagen, a la anotación `@GetMapping` podemos agregar más parámetros separando cada uno con “/”



@PathVariable

Nos permite extraer información que es parte de la estructura de la URI pero que no se trata como un par nombre=valor, a diferencia del @RequestParam que representa un valor que se envía en un pedido.

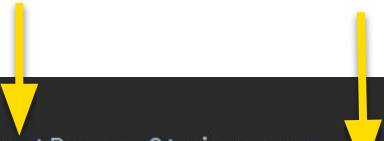
```
@GetMapping("/user/{userId}")
public User getUser(@PathVariable("userId") String userId) {
    ...
}
```

En la práctica, ambas formas se pueden usar y producen el mismo resultado, y muchas veces se usan combinadas.

@Request Param



- En una anotación que permite recibir parámetros desde una ruta mediante el método GET, para trabajar con ellos e incluso poder emitir una respuesta que dependa de los parámetros que sean obtenidos.
- Cada uno de los parámetros generalmente se ubican en la URL después de un signo de pregunta “?” y están anidados por un “&”.
- Por ejemplo: <http://localhost:8080/student?name=Horacio&lastname=Quiroga>



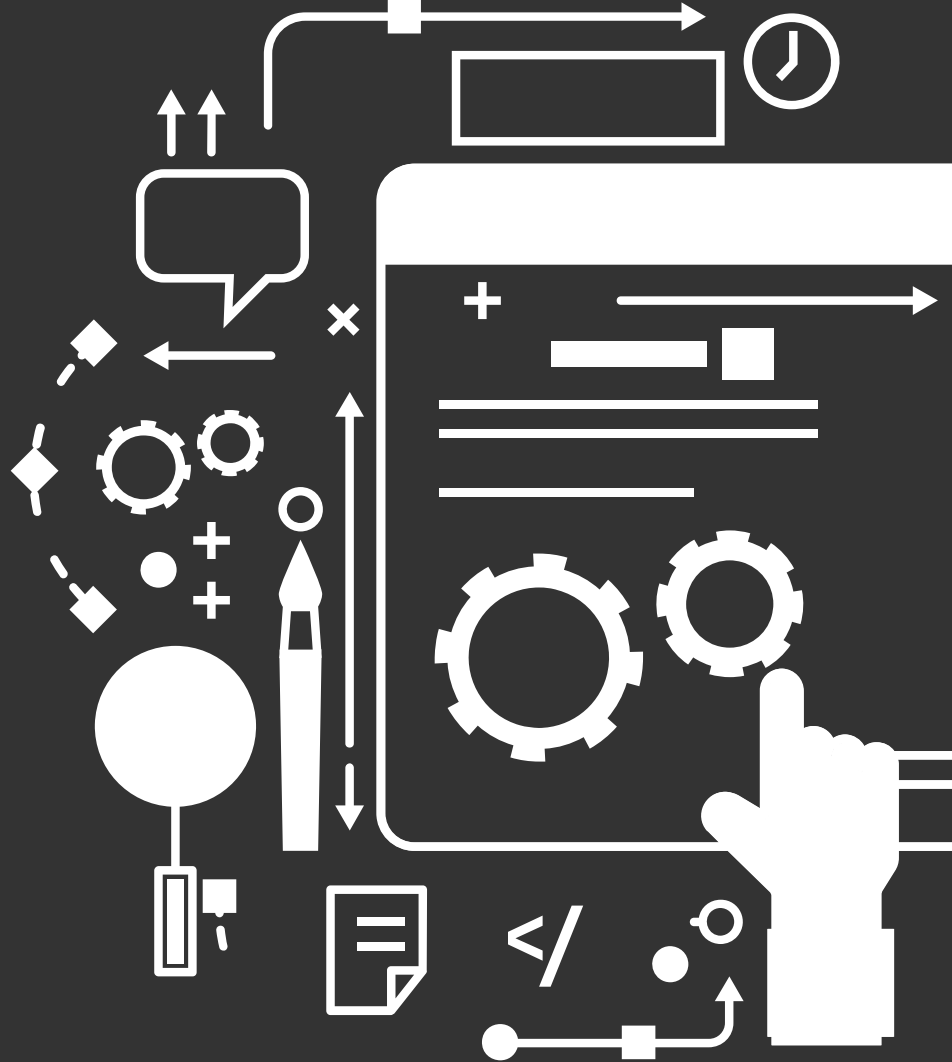
```
@GetMapping (path = "/student/")  
public Student findStudent (@RequestParam String name,  
                             @RequestParam String lastname) {  
  
    return findStudent(name,lastname);  
}
```

JSON

J **A**
S **V**
O **S**
N **C**
O **R**
T **I**
N **P**
O **T**
A **T**
I **O**
N **I**
O **N**

IT BOARDING

BOOTCAMP



// ¿Qué es JSON?

JSON es un **formato de texto sencillo** utilizado para el **intercambio de datos** entre distintos sistemas.

// SINTAXIS

JSON tiene dos elementos centrales los objetos y sus valores intrínsecos:

Objetos

- Están representados mediante llaves, su apertura { indica el comienzo de la estructura de un objeto y su cierre el final } .

Valores Intrínsecos

Los objetos tienen valores / estado y los mismos se representan mediante **claves** (o keys) y **valores** (o value).

- Las Claves deben ser cadenas de caracteres y representan el “nombre” del valor.
- Los valores son tipos de datos soportados por JSON.

Las claves y valores se separan entre sí mediante “:”

En caso de necesitar agregar más claves y valores, se separan entre sí mediante “,”

```
{  
  "clave" : "valor",  
  "clave2" : "valor2",  
  "clave3" : "valor3"  
}
```



Ejemplo

Supongamos que queremos representar a **Homero Simpson** en un **JSON**



```
{  
  "nombre" : "Homero",  
  "apellido" : "Simpson",  
  "hijos" : ["Bart", "Lisa", "Maggie"],  
  "edad" : 40,  
  "esposa" : true  
}
```



{JSON}

JavaScript Object Notation

```
{
  "nombre" : "Gryffindor",
  "colores" : ["Amarillo",
    "Bordó"],
  "integrantes" : [
    { "nombre" : "Harry",
      "apellido" : "Potter"
    },
    { "nombre" : "Hermione",
      "apellido" : "Granger"
    },
    { "nombre" : "Ron",
      "apellido" : "Weasley"
    }
  ]
}
```

Ejemplo 2



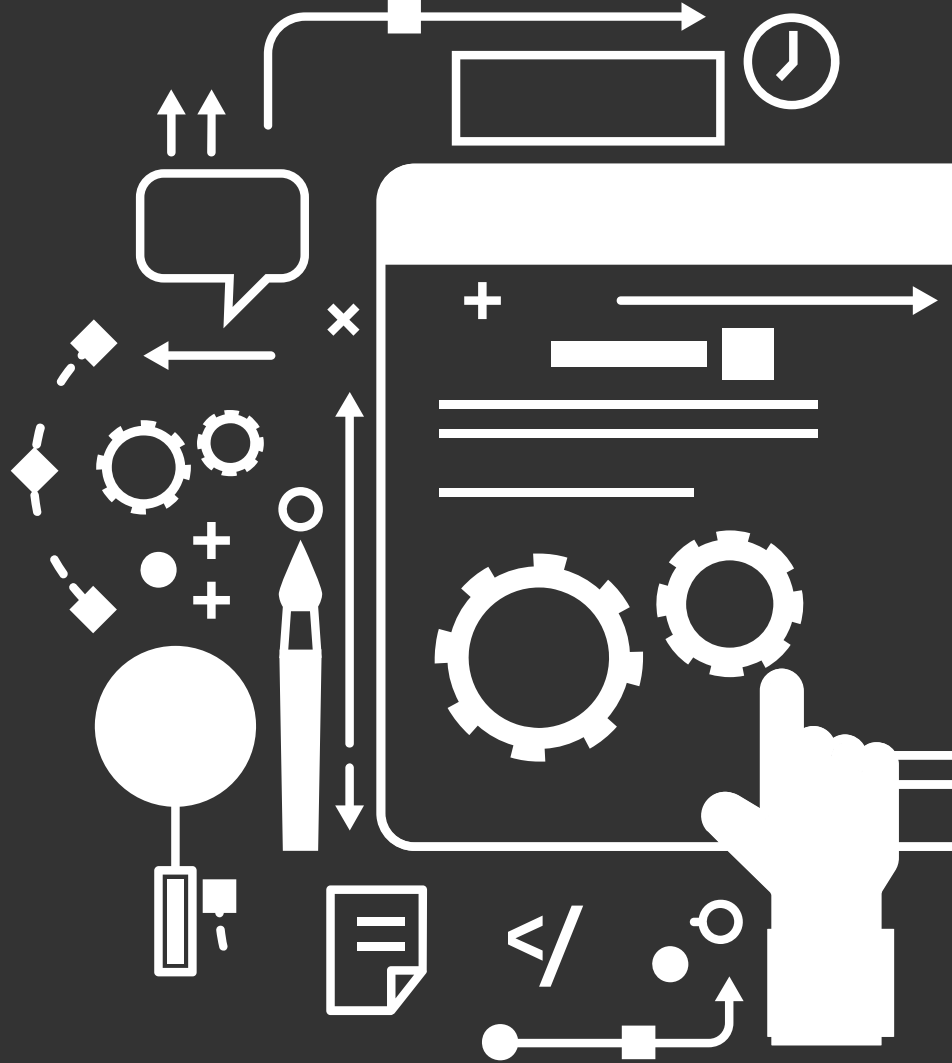
Existen situaciones donde necesitemos representar uno o varios objetos dentro del objeto JSON principal que tengamos. Por ejemplo, supongamos que queremos representar los integrantes de la casa Gryffindor de Harry Potter



Response

IT BOARDING

BOOTCAMP



Respuestas

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.



```
[
  {
    "id":1,
    "titulo":"titulo de la
entrada",
    "autor":"autor de la entrada"
  },
  {
    "id":2,
    "titulo":"titulo de la
segunda entrada",
    "autor":"autor de la segunda
entrada"
  }
]
```

IT BOARDING

BOOTCAMP

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)

Actualicemos la calculadora de edades para que devuelva un DTO en lugar de un String

IT BOARDING

BOOTCAMP

```
public class AgeDTO {  
    private String date;  
    private Integer age;  
  
    public String getDate() {  
        return date;  
    }  
  
    public void setDate(String date) {  
        this.date = date;  
    }  
  
    public Integer getAge() {  
        return age;  
    }  
  
    public void setAge(Integer age) {  
        this.age = age;  
    }  
}
```

@RestController

public class AgeCalculatorRestController {

 @GetMapping("/{day}/{month}/{year}")

 @ResponseBody

 public AgeDTO getAge(@PathVariable Integer day, @PathVariable Integer month, @PathVariable Integer year) {

 AgeDTO ageDTO = new AgeDTO();

 ageDTO.setDate(day + "/" + month + "/" + year);

 ageDTO.setAge(this.calculateAge(day, month, year));

 return ageDTO;

 }

 private Integer calculateAge(Integer day, Integer month, Integer year) {

 Period age;

 try {

 LocalDate date = LocalDate.of(year, month, day);

 age = Period.between(date, LocalDate.now());

 return age.getYears();

 } catch (Exception e) {

 return 0;

 }

 }

}

IT BOARDING

BOOTCAMP

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.



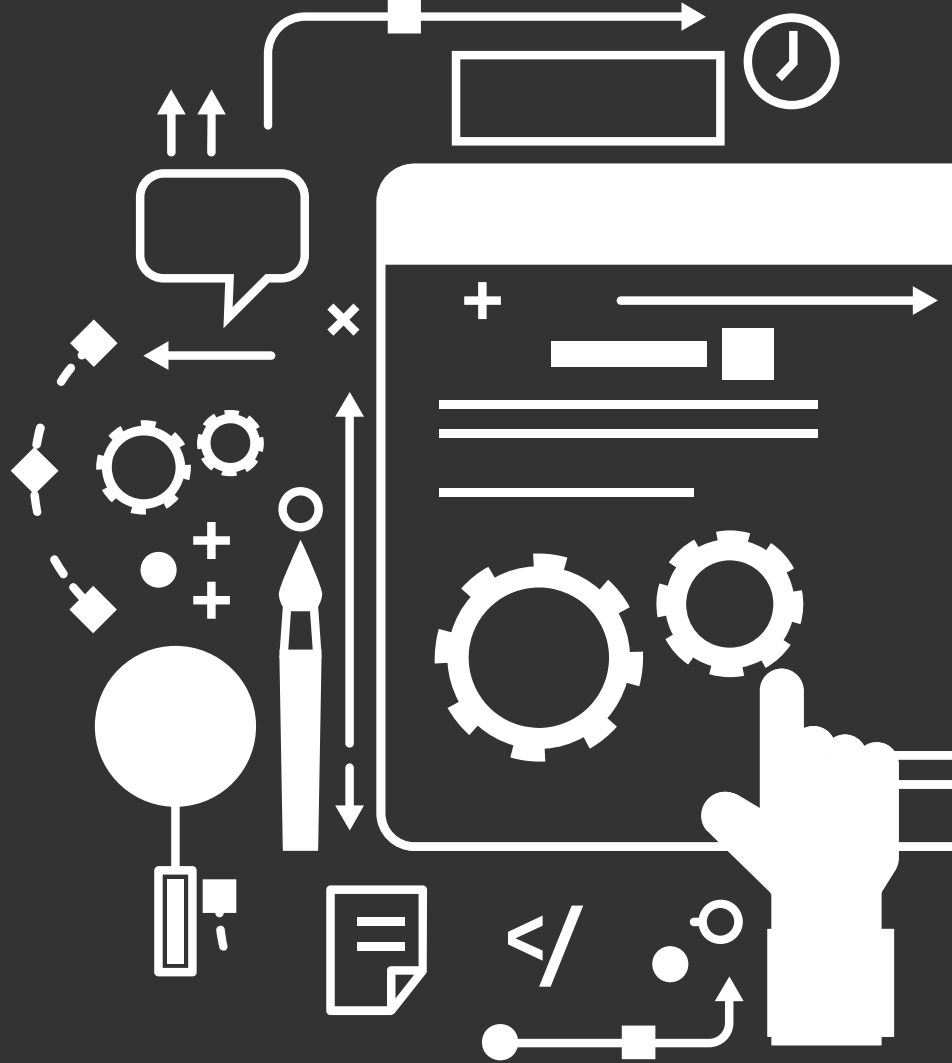
```
@GetMapping("/hello")
ResponseEntity<String> hello() {
    return new ResponseEntity<>("Hello World!", HttpStatus.OK);
}
```



POST

IT BOARDING

BOOTCAMP



¿Cuándo usamos el método HTTP POST?

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) { ... }
```

pring 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 del 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 **obtener información sobre un determinado recurso** sin retornar el registro.

IT BOARDING

BOOTCAMP

