

Arquitecturas de Software para Aplicaciones Empresariales

Clase ResponseEntity

PROGRAMA DE INGENIERIA DE SISTEMAS

Ing. Daniel Eduardo Paz Perafán (danielp@Unicauca.edu.co)

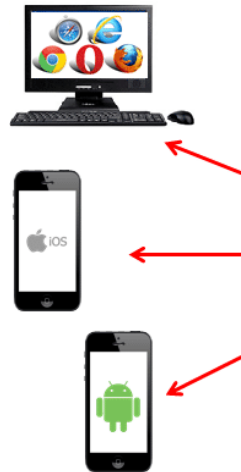
Ing. Pablo A. Magé (pmage@Unicauca.edu.co)



Donde realizar las validaciones

UNIVERSIDAD DEL CAUCA – FIET
DEPARTAMENTO DE SISTEMAS

FRONTEND



By Parzibyte

Agregar producto

Nombre del producto
 ✖
El nombre debe medir entre 1 y 50

Código de barras
 ✖
El código debe medir entre 1 y 50

Existencia actual
 ✖
Debes especificar la existencia

Precio
 ✖
Debes especificar el precio

Guardar Ver todos

© 2019 de ventas Spring Boot API con ❤️ por Parzibyte

BACKEND



BASE DE DATOS



Respuesta

	Versión	Código respuesta
Primera línea	HTTP/1.1	200 OK
Encabezados	Server: wikipedia.org Content-Type: text/html Content-Lenght: 2026	
Cuerpo	<html> ... </html>	

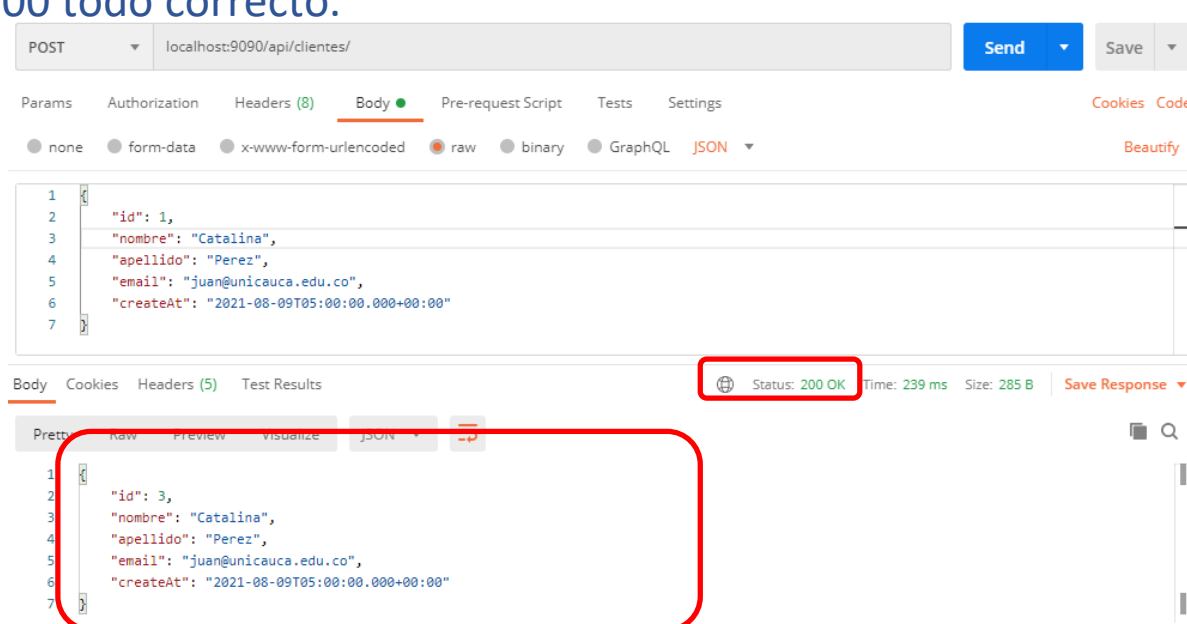
¿Cómo modificamos la respuesta HTTP?

Analizando el ejemplo del servicio que permite crear un cliente, por defecto se retorna un código 200

```
@RestController
@RequestMapping("/api")
public class ClienteRestController {

    @PostMapping("/clientes")
    public Cliente create(@RequestBody Cliente cliente) {
        Cliente objCliente = null;
        objCliente = clienteService.save(cliente);
        return objCliente;
    }
}
```

La petición REST se realiza y obtendremos como resultado que todo ha ido bien . En este caso un código 200 todo correcto.



Todo funciona correctamente pero no es suficiente ya que cuando se realiza una petición POST, se debería recibir un status code de 201 si la petición se ha realizado de forma correcta.

201 Created

El código de respuesta de estado de éxito creado HTTP 201 Created indica que la solicitud ha tenido éxito y ha llevado a la creación de un recurso.

El nuevo recurso se crea efectivamente antes de enviar esta respuesta. y el nuevo recurso se devuelve en el cuerpo del mensaje, su ubicación es la URL de la solicitud o el contenido del encabezado de la Ubicación

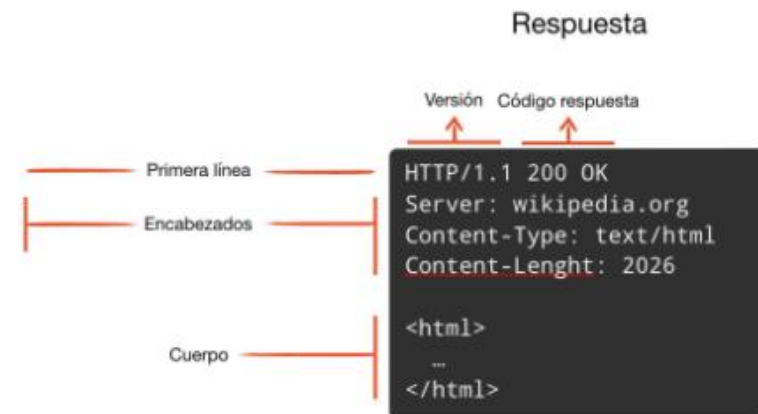
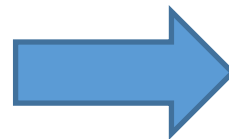
Todo funciona correctamente pero no es suficiente ya que cuando se realiza una petición POST, se debería recibir un status code de 201 si la petición se ha realizado de forma correcta.

Por lo tanto el comportamiento de Spring por defecto no es el más deseado

¿Cómo podemos cambiar este comportamiento? .

Haciendo uso de la clase ResponseEntity que nos permite afinar más las respuestas.

Con ella podemos agregar nuevos encabezados, establecer el cuerpo de la respuesta y el código retornado



La primera línea de un mensaje de respuesta tiene un código de 3 dígitos que le indica al cliente cómo interpretar la respuesta.

Los códigos de respuesta se dividen en cinco categorías dependiendo del dígito con el que inician:

- **1XX**: Información
- **2XX**: Éxito
- **3XX**: Redirección
- **4XX**: Error en el cliente
- **5XX**: Error en el servidor

Estás familiarizado(a) con el famoso error **404** que retornan los servidores cuando el recurso no fue encontrado. O con el error **500** cuando ocurre un error en el servidor. Pero existen muchos más.

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

Si queremos utilizar esta clase, tenemos que devolver un objeto de ella desde el punto final y Spring se encarga del resto.

ResponseEntity es un tipo genérico. En consecuencia, podemos usar cualquier tipo como cuerpo de respuesta:

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

HttpStatus.**OK** representa el código 200

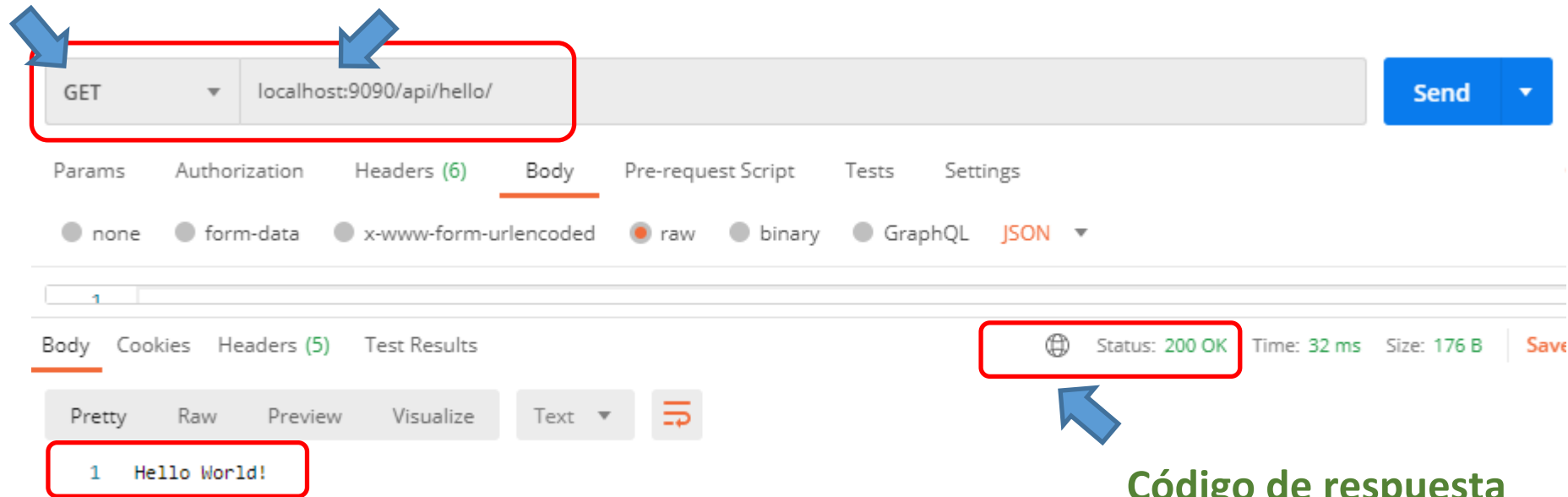
HttpStatus.**NOT_FOUND** representa el código 404

HttpStatus.**INTERNAL_SERVER_ERROR** representa el código 500

Al consumir el servicio mediante postman podemos ver la respuesta con código 200

Método

Ruta



Código de respuesta

Cuerpo de la respuesta

Al consumir el servicio mediante postman podemos ver la respuesta con código 400

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

The screenshot shows the Postman interface. On the left, the request is a GET to `localhost:8085/api/hello`. The 'Query' tab is selected, showing a table for query parameters with columns 'parameter' and 'value'. On the right, the response is displayed with a status of '400 Bad Request', size of '12 Bytes', and time of '14 ms'. The response body is 'Hello World!'.

parameter	value

Status: 400 Bad Request Size: 12 Bytes Time: 14 ms

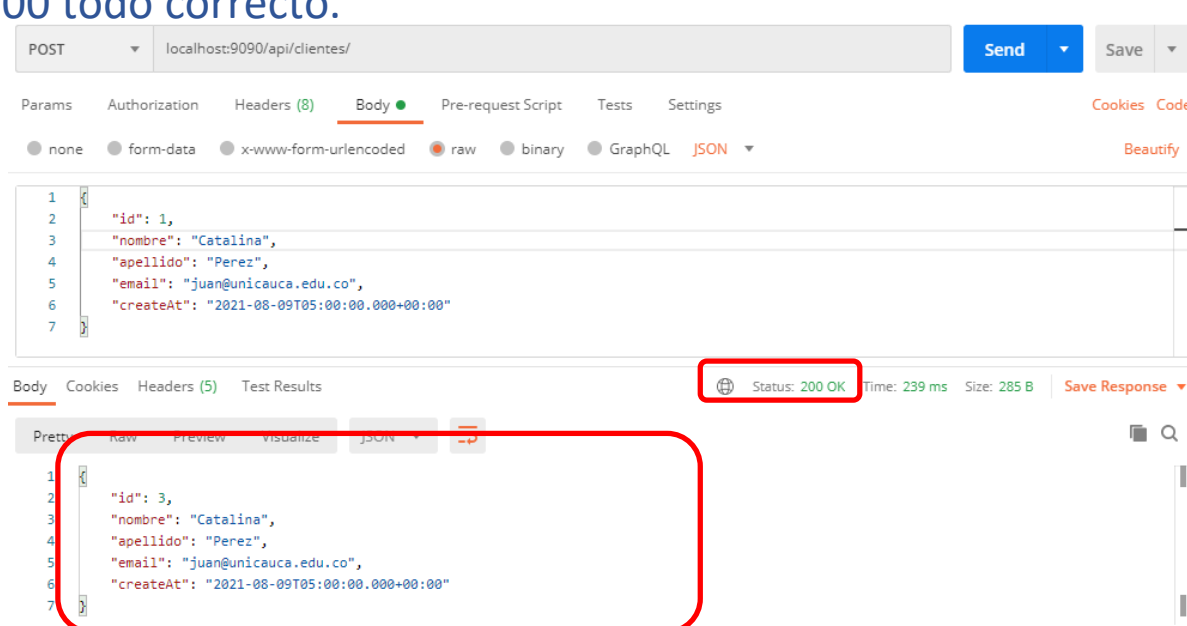
Response Headers 4 Cookies Results Docs {} ≡

1 Hello World!

Analizando el ejemplo del servicio que permite crear un cliente, por defecto se retorna un código 200

```
@PostMapping("/clientes")
public Cliente create(@RequestBody Cliente cliente) {
    Cliente objCliente = null;
    objCliente = clienteService.save(cliente);
    return objCliente;
}
```

La petición REST se realiza y obtendremos como resultado que todo ha ido bien . En este caso un código 200 todo correcto.



De la siguiente forma cuando nosotros gestionemos la petición al servicio REST realizando una petición POST este servicio nos devolverá un nuevo resultado.

```
@PostMapping("/clientes")
public ResponseEntity<?> create(@RequestBody Cliente cliente) {
    Cliente objCliente = null;
    objCliente = clienteService.save(cliente);
    ResponseEntity<Cliente> objRespuesta= new ResponseEntity<Cliente>(objCliente, HttpStatus.CREATED);
    return objRespuesta;
}
```

En este caso el diseñador del API ha decidido que como respuesta de una petición de inserción se devuelve el registro insertado así como un nuevo código HTTP de 201 que significa que un recurso nuevo ha sido creado.

Crear cliente

The screenshot displays a REST client interface with the following components:

- Request Section:**
 - Method: **POST**
 - URL: `localhost:9090/api/clientes/`
 - Buttons: **Send** and **Save**
 - Tabs: Params, Authorization, Headers (8), **Body** (selected), Pre-request Script, Tests, Settings
 - Content Type: **JSON** (selected from a dropdown menu)
- Request Body:**

```
1 {  
2   "nombre": "Andres",  
3   "apellido": "Lopez",  
4   "email": "andres@unicauca.edu.co",  
5   "createAt": "2021-08-09T05:00:00.000+00:00"  
6 }
```
- Response Section:**
 - Tabs: **Body** (selected), Cookies, Headers (5), Test Results
 - Status: **201 Created** (highlighted with a red box)
 - Time: 242 ms, Size: 290 B
 - Buttons: **Save Response**
 - View Options: Pretty, Raw, Preview, Visualize, **JSON** (selected)
- Response Body:**

```
1 {  
2   "id": 4,  
3   "nombre": "Andres",  
4   "apellido": "Lopez",  
5   "email": "andres@unicauca.edu.co",  
6   "createAt": "2021-08-09T05:00:00.000+00:00"  
7 }
```

Se puede utilizar un HashMap para devolver una o varias respuestas. Cada respuesta tendrá un identificador y una descripción.

```
@GetMapping("/clientes/{id}")
public ResponseEntity<?> show(@PathVariable Integer id) {
    Cliente objCliente = null;
    HashMap<String, Object> respuestas= new HashMap();
    ResponseEntity<?> objRespuesta;
    objCliente = clienteService.findById(id);
    if(objCliente==null)
    {
        respuestas.put("mensaje", "El cliente con ID: "+id+" no existe en la base de datos");
        objRespuesta= new ResponseEntity<HashMap<String, Object>>(respuestas, HttpStatus.NOT_FOUND);
    }
    else
    {
        objRespuesta= new ResponseEntity<Cliente>(objCliente, HttpStatus.OK);
    }

    return objRespuesta;
}
```

Error al intentar consultar un cliente que no existe

The screenshot displays a REST client interface with the following components:

- Request Bar:** Method `GET`, URL `localhost:8085/api/clientes/2`, and a `Send` button.
- Request Tabs:** `Query` (selected), `Headers` (2), `Auth`, `Body`, `Tests`, and `Pre Run` (New).
- Query Parameters:** A table with columns `parameter` and `value`, currently empty.
- Response Bar:** Status `404 Not Found`, Size `64 Bytes`, and Time `12 ms`. It includes control icons for expand, pause, refresh, download, upload, redo, and close.
- Response Tabs:** `Response` (selected), `Headers` (4), `Cookies`, `Results`, and `Docs`.
- Response Body:** A JSON object:

```
{  "mensaje": "El cliente con ID: 2 no existe en la base de datos"}
```

Es posible también capturar un error generado al intentar conectarse a una base de datos

```
@GetMapping("/clientes/{id}")
public ResponseEntity<?> show(@PathVariable Integer id) {
    Cliente objCliente = null;
    HashMap<String, Object> respuestas= new HashMap();
    ResponseEntity<?> objRespuesta;
    try{
        objCliente = clienteService.findById(id);

        if(objCliente==null)
        {
            respuestas.put("mensaje", "El cliente con ID: "+id+" no existe en la base de datos");
            objRespuesta= new ResponseEntity<HashMap<String, Object>>(respuestas,HttpStatus.NOT_FOUND);
        }
        else
        {
            objRespuesta= new ResponseEntity<Cliente>(objCliente,HttpStatus.OK);
        }
    }
    catch(DataAccessException e)
    {
        respuestas.put("mensaje", "Error al realizar la consulta en la base de datos");
        respuestas.put("descripción del error", e.getMessage());
        objRespuesta= new ResponseEntity<HashMap<String, Object>>(respuestas,HttpStatus.INTERNAL_SERVER_ERROR);
    }
    return objRespuesta;
}
```

Al capturar los posibles errores, debemos considerar las restricciones propias de la tabla

```
@Entity
@Table(name="Clientes")
public class Cliente {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    @Column(nullable = false)
    private String nombre;
    @Column(nullable = false)
    private String apellido;
    @Column(nullable = false, unique = true)
    private String email;
    private Date createdAt;
```


En el API REST podemos capturar un error asociado al intentar registrar un cliente que no cumple con una restricción, por ejemplo campos obligatorios, campos únicos, formatos establecidos.

```
@PostMapping("/clientes")
public ResponseEntity<?> create(@RequestBody Cliente cliente) {
    Cliente objCliente = null;
    HashMap<String, Object> respuestas= new HashMap();
    ResponseEntity<?> objRespuesta;
    try
    {
        objCliente = clienteService.save(cliente);
        objRespuesta= new ResponseEntity<Cliente>(objCliente,HttpStatus.CREATED);
    }
    catch(DataAccessException e)
    {
        respuestas.put("mensaje", "Error al realizar la inserción en la base de datos");
        respuestas.put("descripción del error", e.getMessage());
        objRespuesta= new ResponseEntity<HashMap<String, Object>>(respuestas,HttpStatus.BAD_REQUEST);
    }

    return objRespuesta;
}
```

Es posible también capturar un error al intentar registrar un cliente sin un apellido, el cual es un campo obligatorio

POST localhost:8085/api/clientes

Untitled Request

POST localhost:8085/api/clientes

Send Save

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Code

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2   "nombre": "Catalina",
3   "createAt": "2021-08-09T05:00:00.000+00:00"
4 }
```

Body Cookies Headers (4) Test Results

Status: 400 Bad Request Time: 1352 ms Size: 547 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "descripción del error": "not-null property references a null or transient value : co.edu.unicauca.validacion_errores_back.core.models.Cliente.apellido; nested exception is org.hibernate.PropertyValueException: not-null property references a null or transient value : co.edu.unicauca.validacion_errores_back.core.models.Cliente.apellido",
3   "mensaje": "Error al realizar la inserción en la base de datos"
4 }
```

Es posible también capturar un error al intentar registrar un cliente con un correo repetido

The screenshot displays a REST client interface with the following details:

- Request:** Method `POST`, URL `localhost:8085/api/clientes`. The request body is a JSON object: `{ "nombre": "Adriana", "apellido": "Lopez", "email": "andres@unicauca.edu.co", "createAt": "2021-08-09T05:00:00.000+00:00" }`.
- Response:** Status `400 Bad Request`, Time `1441 ms`, Size `397 B`. The response body is a JSON object: `{ "descripción del error": "could not execute statement; SQL [n/a]; constraint [null]; nested exception is org.hibernate.exception.ConstraintViolationException: could not execute statement", "mensaje": "Error al realizar la inserción en la base de datos" }`.

Red boxes highlight the request body and the error response body.

En el API REST podemos capturar un error asociado al intentar eliminar un cliente que no existe

```
@DeleteMapping("/clientes/{id}")
public ResponseEntity<?> delete(@PathVariable Integer id) {
    Cliente objCliente = null;
    HashMap<String, Object> respuestas = new HashMap();
    ResponseEntity<?> objRespuesta;
    try {
        objCliente = clienteService.findById(id);

        if (objCliente == null) {
            respuestas.put(key: "mensaje",
                "El cliente con ID: " + id + "que se desea eliminar no existe en la base de datos");
            objRespuesta = new ResponseEntity<HashMap<String, Object>>(respuestas, HttpStatus.NOT_FOUND);
        } else {
            clienteService.delete(id);
            objRespuesta = new ResponseEntity<Cliente>(objCliente, HttpStatus.OK);
        }
    } catch (DataAccessException e) {
        respuestas.put(key: "mensaje", value: "Error al realizar la eliminación del cliente en la base de datos");
        respuestas.put(key: "descripción del error", e.getMessage());
        objRespuesta = new ResponseEntity<HashMap<String, Object>>(respuestas, HttpStatus.INTERNAL_SERVER_ERROR);
    }

    return objRespuesta;
}
```

Eliminar cliente

En el API REST podemos capturar un error asociado al intentar eliminar un cliente que no existe

The screenshot displays an API client interface with a request and response view. The request is a DELETE method to the endpoint `localhost:8085/api/clientes/2`. The response is a 404 Not Found status with a size of 85 Bytes and a time of 15 ms. The response body contains a JSON object with a message in Spanish: `{ "mensaje": "El cliente con ID: 2 que se desea eliminar no existe en la base de datos" }`.

Method	URL	Status	Size	Time
DELETE	localhost:8085/api/clientes/2	404 Not Found	85 Bytes	15 ms

Query Parameters

parameter	value

Response

```
1 {
2   "mensaje": "El cliente con ID: 2 que se desea eliminar no existe
3             en la base de datos"
}
```

En el FRONTEND construido en angular, la respuesta HTTP construida mediante la clase ResponseEntity es analizada, y a partir de su código de error se puede detectar si ocurrió un error en la operación.

throwError

Crea un Observable que no emite elementos al Observador e inmediatamente emite una notificación de error. Necesitamos importarlo desde:

```
import {throwError} from 'rxjs';

create(cliente: Cliente) : Observable<Cliente> {
  return this.http.post<Cliente>(this.urlEndPoint, cliente, {headers: this.httpHeaders}).pipe(
    catchError(
      e => {
        console.log(e.error.mensaje);
        swal.fire('Error al crear el cliente', e.error.mensaje, 'error');
        return throwError(e);
      }
    )
  );
}
```

Manejar error en el front

En el FRONTEND construido en angular, un objeto de tipo ResponseEntity es analizado, y a partir de su código de error se puede detectar si ocurrió un error en la operación.

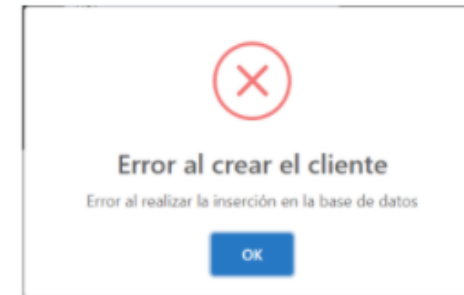
Campo nulo

Crear cliente

Nombre

Apellido

Email



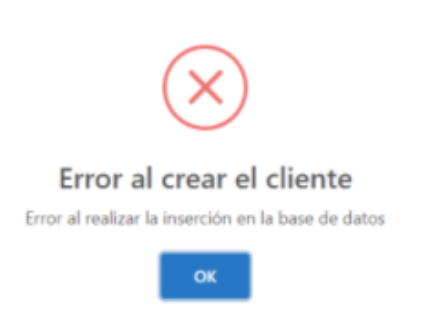
Correo duplicado

Crear cliente

Nombre

Apellido

Email



Podemos establecer una cabecera de la respuesta haciendo uso de la clase HttpHeaders:

```
@GetMapping("/cabecera/{id}")
ResponseBody<String> cabeceraPersonalizada(@PathVariable Integer id) {

    HttpHeaders cabecera = new HttpHeaders();
    cabecera.add("Estado Cliente", "Cliente con id " + id + ": habilitado");

    return new ResponseEntity<>("Bienvenido " + id, cabecera, HttpStatus.OK);
}
```

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** localhost:9090/api/cabecera/1/
- Buttons:** Send, Save
- Tabs:** Params, Authorization, Headers (6), Body, Pre-request Script, Tests, Settings, Cookies, Code
- Form Data:** none, form-data, x-www-form-urlencoded, raw, binary, GraphQL, JSON
- Body:** 1
- Response:** Status: 200 OK, Time: 34 ms, Size: 222 B, Save Response
- Headers Table:**

KEY	VALUE
Estado Cliente ⓘ	Cliente con id 1: habilitado
Content-Type ⓘ	text/plain; charset=UTF-8
Content-Length ⓘ	12
Date ⓘ	Sat, 11 Dec 2021 11:31:49 GMT
Keep-Alive ⓘ	timeout=60
Connection ⓘ	keep-alive

Muchas gracias
Preguntas

