Arquitecturas de Software para Aplicaciones Empresariales

Clase ResponseEntity

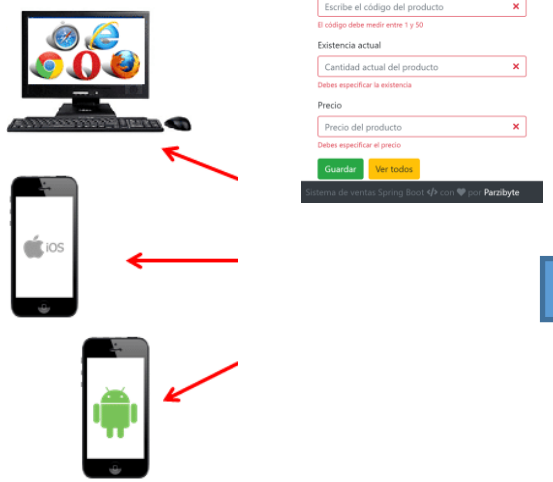
PROGRAMA
DE INGENIERIA DE SISTEMAS

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

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

Donde realizar las validaciones

FRONTEND



BACKEND

BASE DE DATOS

¿Cómo modificamos la respuesta HTTP?

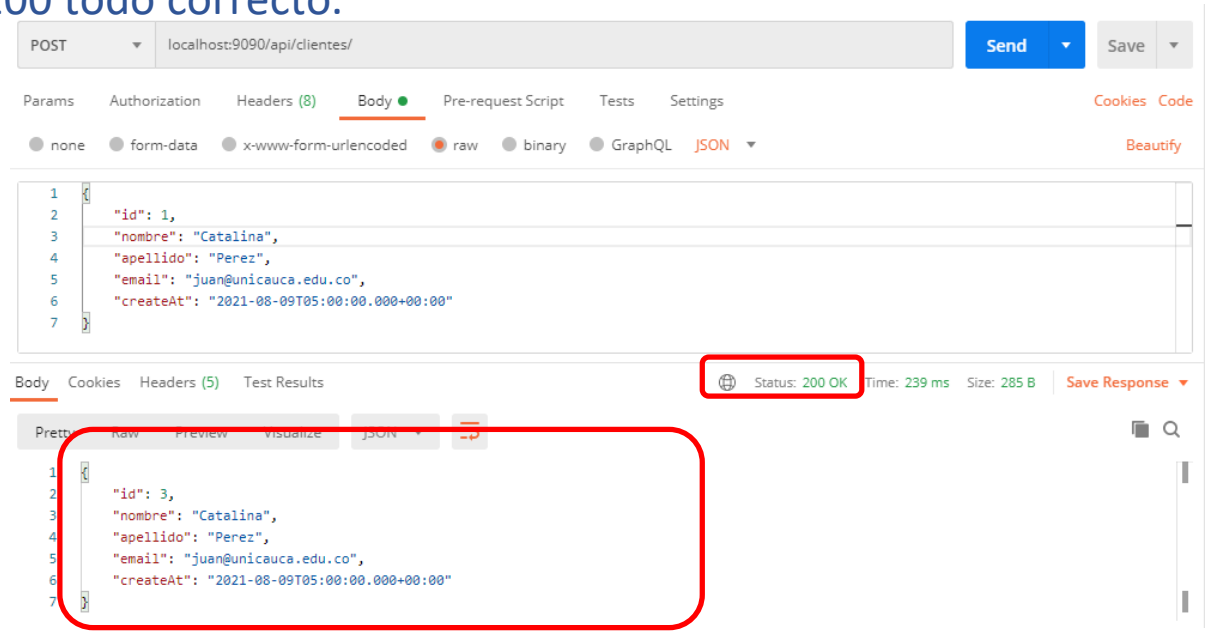
Clase ResponseEntity

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.



Clase ResponseEntity

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 Ubicacion

Clase ResponseEntity

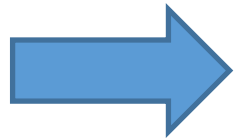
Todo funciona correctamente pero no es suficiente ya que cuando se realiza una petición POST, se debería recibir un status code de 201 si la petición se ha realizado de forma correcta.

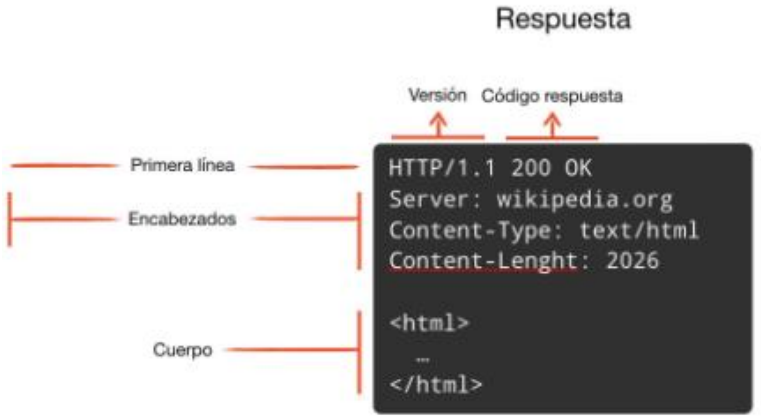
Por lo tanto el comportamiento de Spring por defecto no es el más deseado

¿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





Códigos de respuesta en HTTP

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.

Uso de la clase RespondeEntity

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.

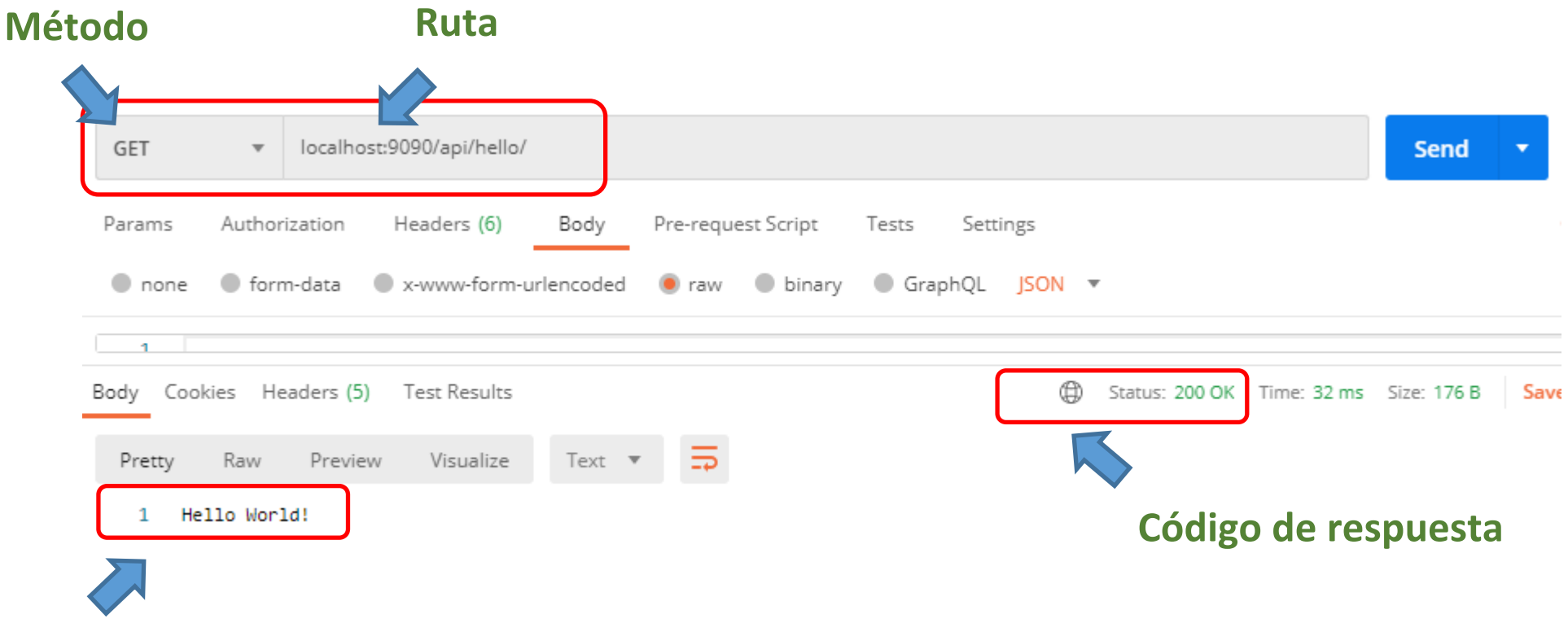
ResponseEntity es un tipo genérico. En consecuencia, podemos usar cualquier tipo como cuerpo de respuesta:

```
@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

Pruebas del servicio

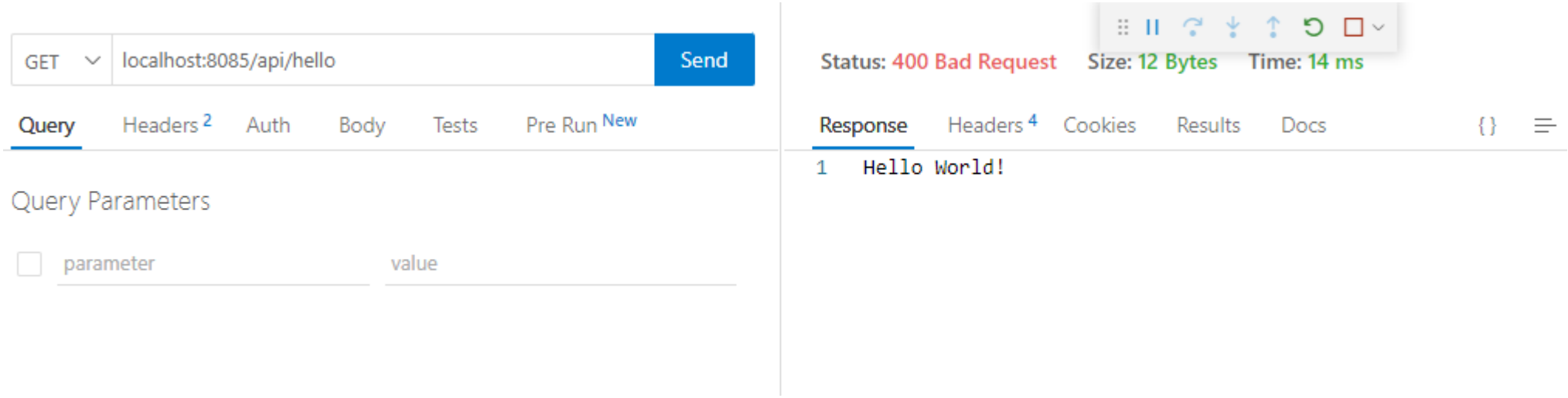
Al consumir el servicio mediante postman podemos ver la respuesta con código 200



Pruebas del servicio

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

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

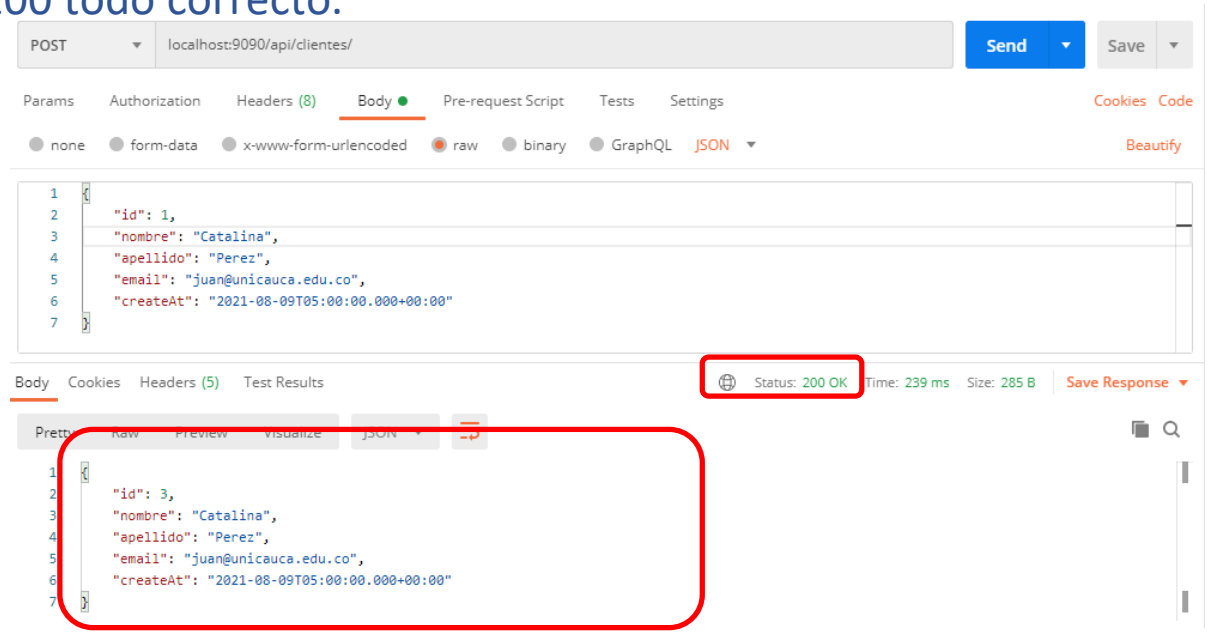


Crear cliente

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.



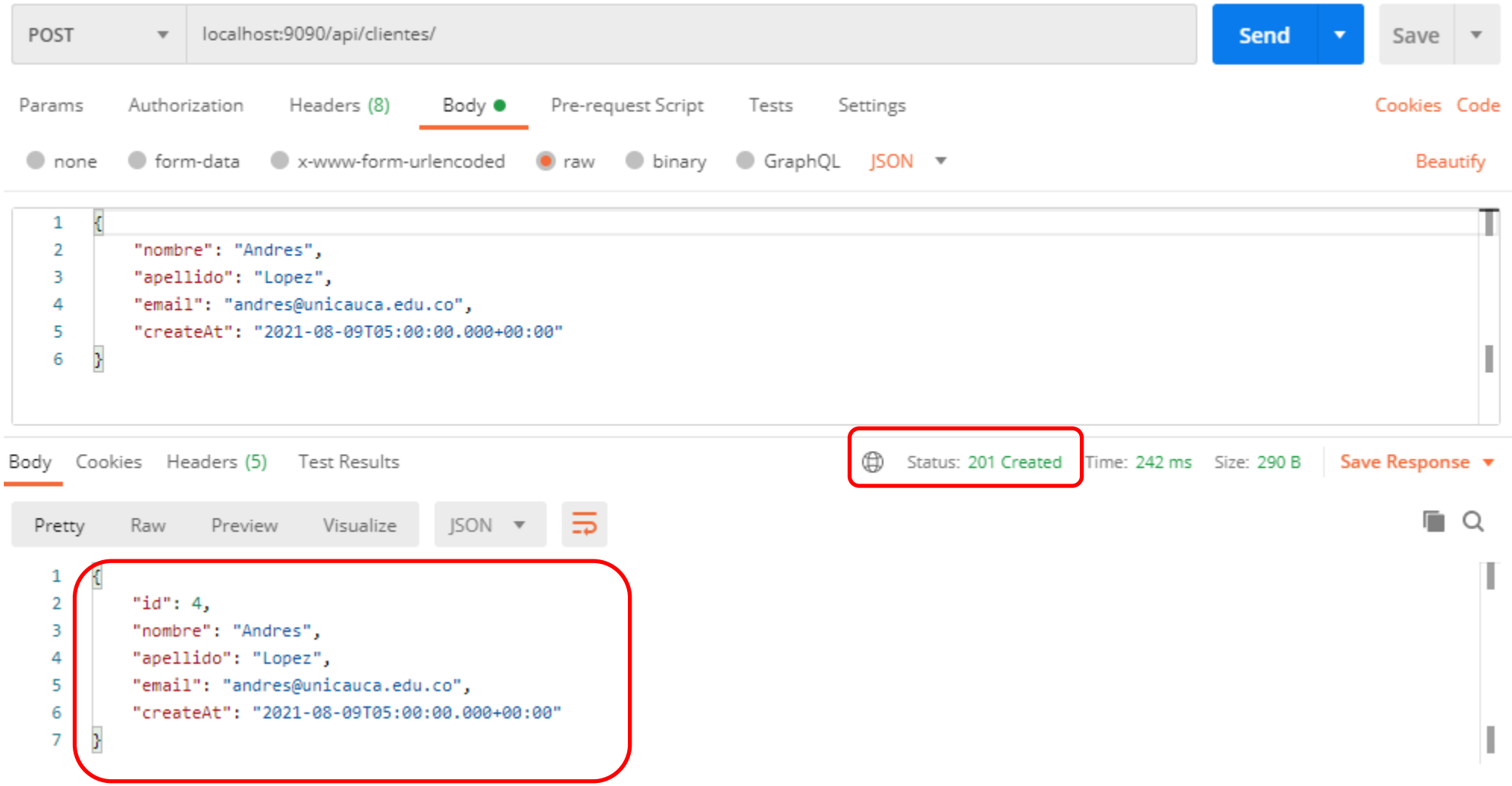
Crear cliente

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



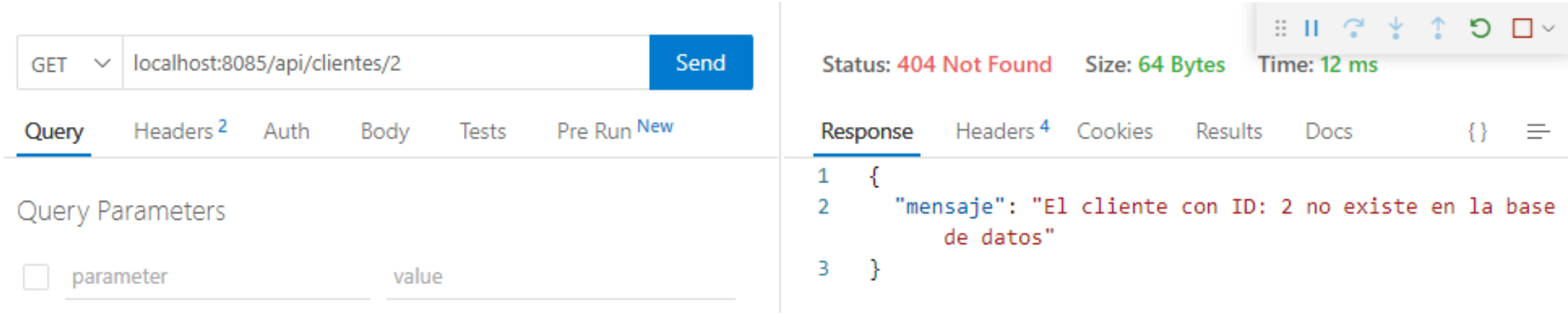
Consultar cliente

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;
    }
```

Consultar cliente

Error al intentar consultar un cliente que no existe



Consultar cliente

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;
  }
```

Crear cliente

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 createAt;
```

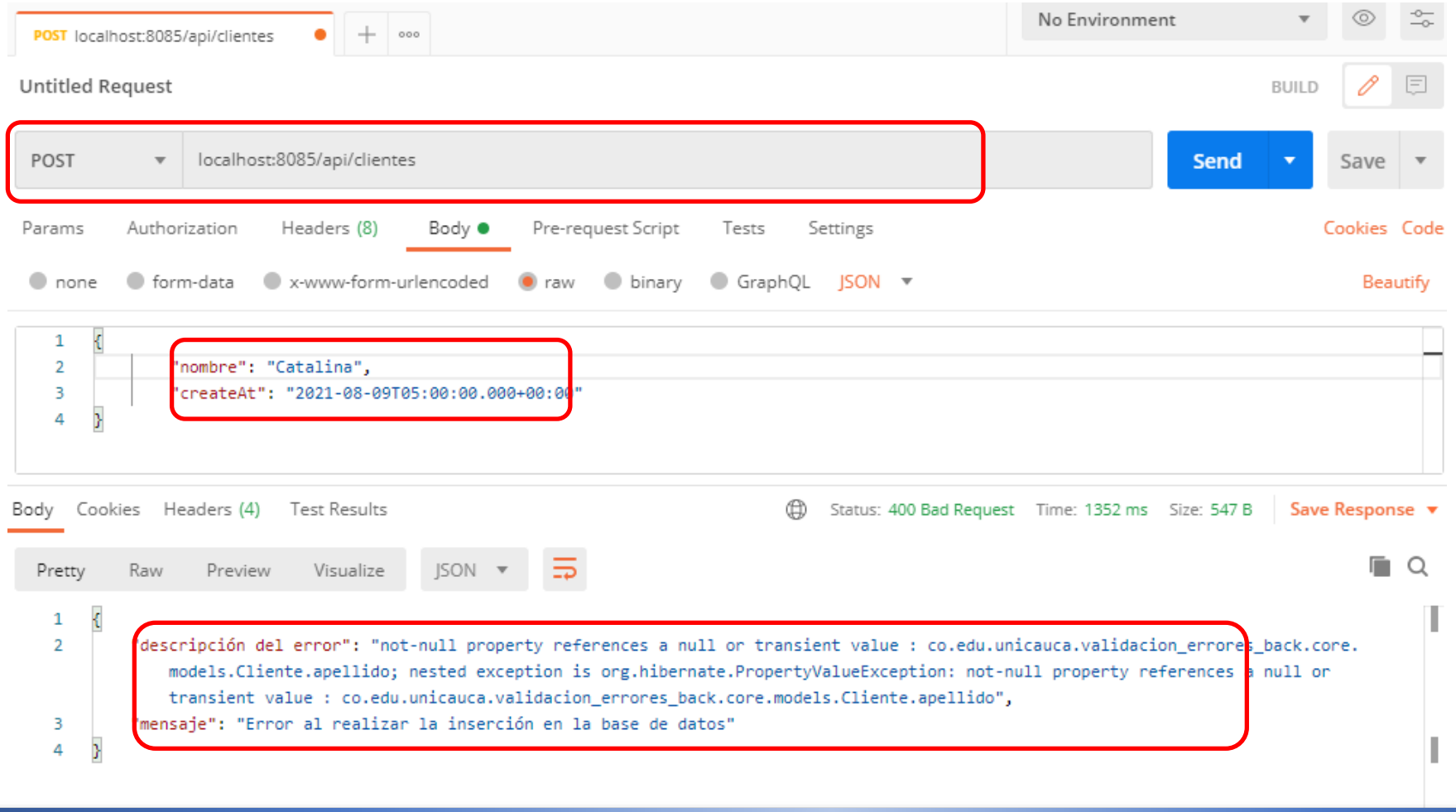
Crear cliente

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;
}
```

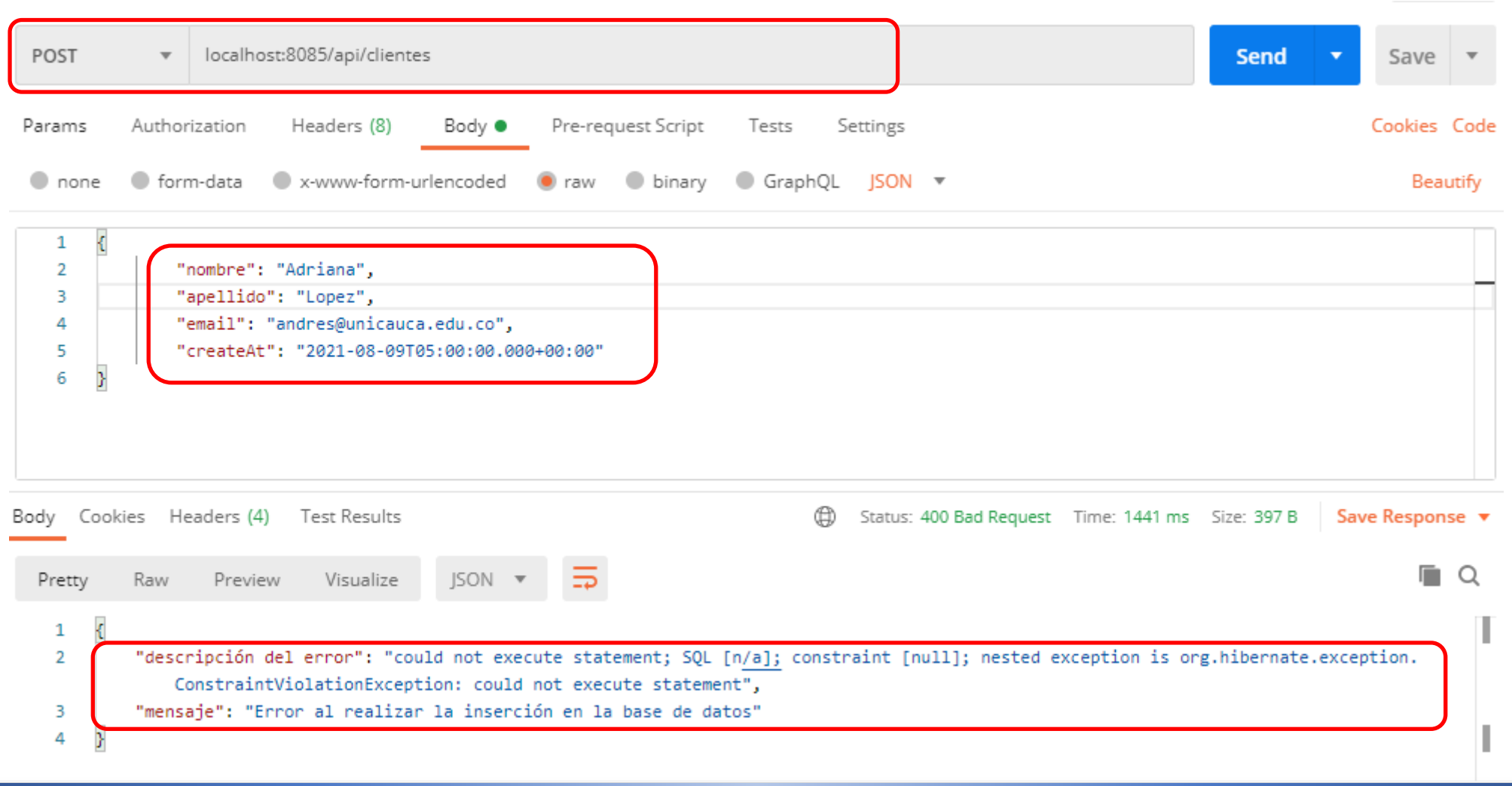
Crear cliente

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



Crear cliente

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



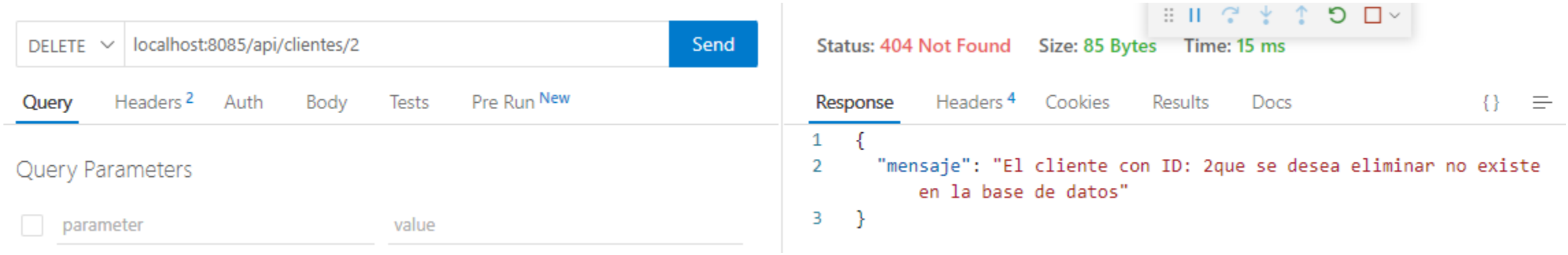
Eliminar cliente

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



Manejar error en el front

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';
```

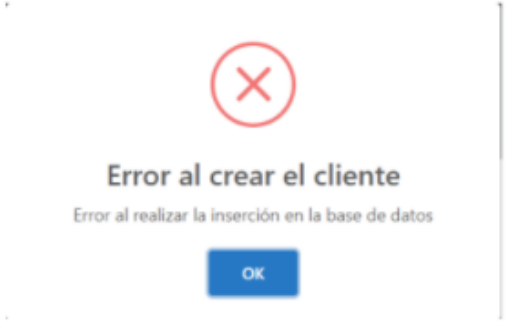
```
create(cliente: Cliente) : Observable<Cliente> {
    return this.http.post<Cliente>(this.urlEndPoint, cliente, {headers: this.httpHeaders}).pipe(
      catchError(
        e => {
            console.log(e.error.mensaje);
            swal.fire('Error al crear el cliente', e.error.mensaje, 'error');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

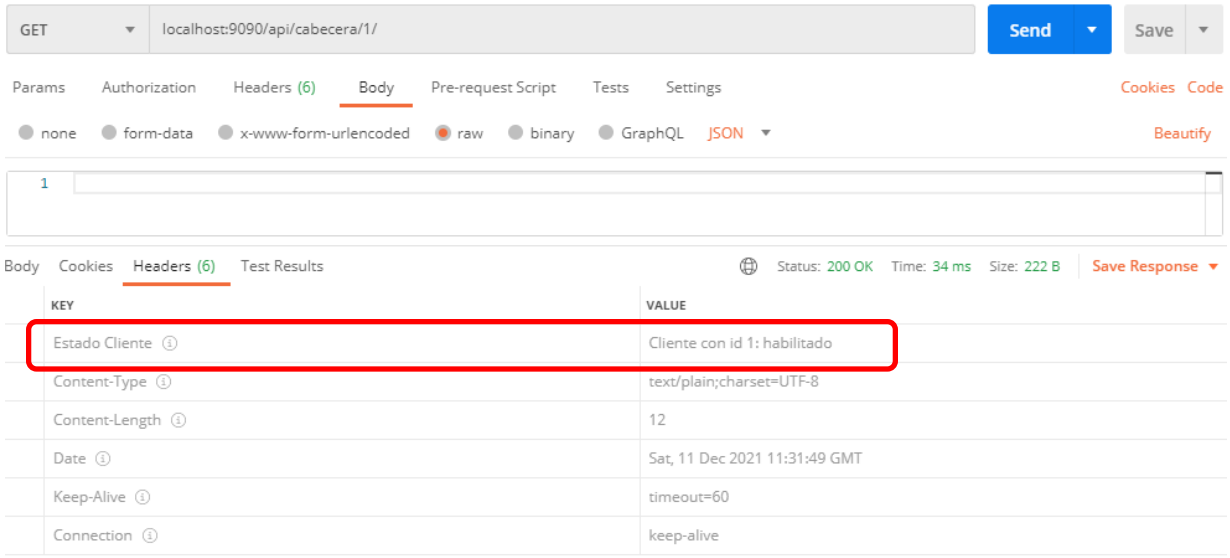
Correo duplicado



Manejar error en el front

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

```
@GetMapping("/cabecera/{id}")
    ResponseEntity<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);
    }
```



Muchas gracias

Preguntas