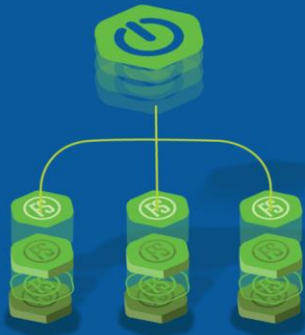


# SPRING BOOT MICROSERVICES ARCHITECTURE

FULL SCALE



AD-4. Tarea en equipo. Microservicios web con Spring Boot

## DESCRIPCIÓN BREVE

Una Empresa tiene recogido en una base de datos los pedidos solicitados por Clientes y el Comercial asociado a la venta. La tabla pedidos recoge el importe total del pedido realizado. La aplicación la van a usar los jefes de comerciales para ver la información, tanto de clientes como de sus comerciales.

Raúl García – Álvaro Bustos

**Desarrollo Web en Entorno Servidor:  
MEAN & Full Stack**

# Contenido

Introducción: ..... 2

Requisitos: ..... 2

Instalación: ..... 2

Configuración: ..... 3

Uso: ..... 4

Código: ..... 9

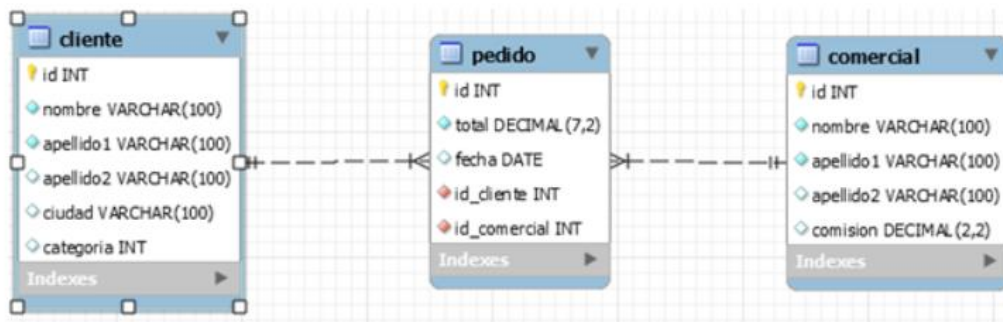
Conclusiones ..... 13

## Introducción:

Una Empresa tiene recogido en una base de datos los pedidos solicitados por Clientes y el Comercial asociado a la venta. La tabla pedidos recoge el importe total del pedido realizado.

La aplicación la van a usar los jefes de comerciales para ver la información, tanto de clientes como de sus comerciales.

Tablas:



## Requisitos:

Para poder ejecutar esta aplicación tenemos que tener el JDK para JavaSE-17, las spring tools en nuestro eclipse o en su defecto el IDE de spring, spring toolSuit, cuando carguemos el proyecto, deberemos actualizarlo, para que nos cargue las diferentes dependencias para su funcionamiento, también tendremos que tener instalado MySQL workbench 8.0 para poder cargar el script(que será adjuntado con esta documentación), y asegurarse del puerto en el que estará desplegada nuestra aplicación, siempre podemos cambiarlo desde el application.properties.

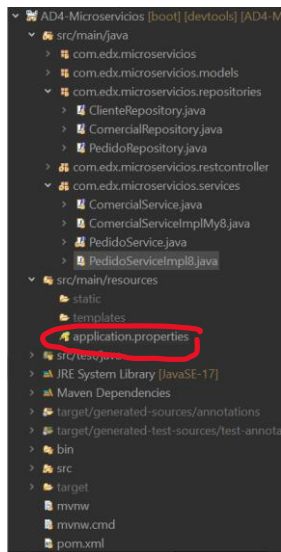
## Instalación:

Para instalar la aplicación, usando eclipse, deberemos importar un proyecto Maven existente y buscar, donde le hayamos guardado, a continuación, como hemos mencionado anteriormente, deberemos actualizar el proyecto para poder cargar todas las dependencias.

Teniendo el dashboard de springboot, simplemente nos quedaría levantar el servidor, y comprobar su funcionamiento con postman, o cualquier programa que nos permita hacer peticiones HTTP.

## Configuración:

En el archivo que encontraremos en src/main/resources llamado application.properties



Vamos a tener esta configuración

```
1 server.port = 8083
2
3 # para mysql 8 bbdd "" con username and password, y driver de mysql 8
4 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
5 spring.datasource.url=jdbc:mysql://localhost:3306/ventasbbdd_2023
6 spring.datasource.username=uventas_2023
7 spring.datasource.password=uventas
8 # para hibernate: no genere las tablas, dialecto de base de datos y estrategia de nombrado de clases
9 # y atributos standard.
10 spring.jpa.generate-ddl=false
11 spring.jpa.show-sql=true
12 spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQLDialect
```

Aquí podremos modificar el puerto donde estará desplegada la aplicación, cambiando el server.port.

También nos encontramos con el nombre del driver, en este caso es mysql, a continuación podríamos modificar la url donde tenemos alojada la bbdd, en este caso la tenemos en localhost y se llama ventasdbbddd\_2023, seguido, pondríamos el nombre de usuario y su contraseña, si es que tuviera.

Después hay una serie de configuración, que en este caso ponemos que no genere las tablas en las bbdd, ya que están creadas, podría hacerse de manera inversa, creando las clases con anotaciones JPA en nuestra clase entities(o beans) y poder persistir en la bbdd, creandolas, mediante un update.

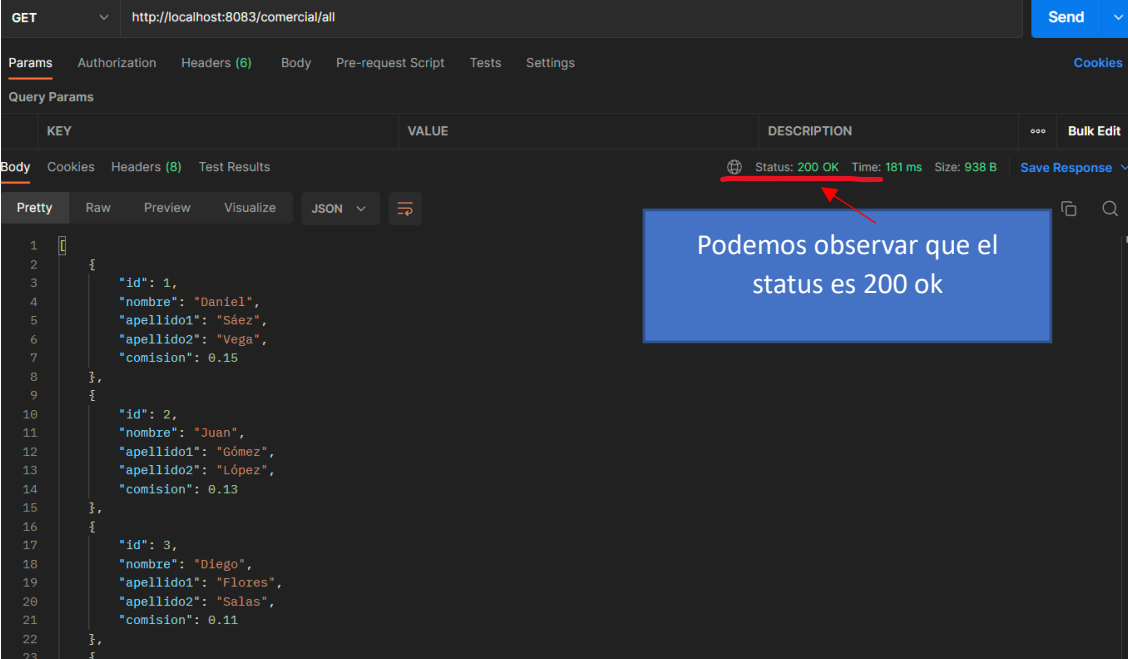
Ponemos que nos muestre las instrucciones SQL que se realizan, con el show-sql.

Y por último añadimos el dialecto mediante el cual se va a intercambiar la información, en este caso es MySQLDialect.

## Uso:

Ahora vamos a mostrar, el funcionamiento de la aplicación con el uso de Postman, vamos a realizar diferentes llamadas a nuestro servicio rest, tanto GET, POST, y DELETE y mostraremos con capturas el resultado.

Primero con método GET vamos a pedir que muestre todos los comerciales, usando este endpoint, <http://localhost:8083/comercial/all>



Podemos observar que el status es 200 ok

```
1 {
2   {
3     "id": 1,
4     "nombre": "Daniel",
5     "apellido1": "Sáez",
6     "apellido2": "Vega",
7     "comision": 0.15
8   },
9   {
10    "id": 2,
11    "nombre": "Juan",
12    "apellido1": "Gómez",
13    "apellido2": "López",
14    "comision": 0.13
15  },
16  {
17    "id": 3,
18    "nombre": "Diego",
19    "apellido1": "Flores",
20    "apellido2": "Salas",
21    "comision": 0.11
22  },
23  }
```

```
36 },
37 {
38   "id": 6,
39   "nombre": "Manuel",
40   "apellido1": "Dominguez",
41   "apellido2": "Hernández",
42   "comision": 0.13
43 },
44 {
45   "id": 7,
46   "nombre": "Antonio",
47   "apellido1": "Vega",
48   "apellido2": "Hernández",
49   "comision": 0.11
50 },
51 {
52   "id": 8,
53   "nombre": "Alfredo",
54   "apellido1": "Ruiz",
55   "apellido2": "Flores",
56   "comision": 0.05
57 }
58 }
```

Podemos ver que nos muestra el resultado de todos los comerciales que tenemos dados de alta, el último en este caso sería el comercial con id: 8,

Usando el método GET, también vamos a realizar una búsqueda de comercial por ID, con este endpoint, <http://localhost:8083/comercial/uno/5>

GET <http://localhost:8083/comercial/uno/5>

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

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (8) Test Results

Status: 302 Found Time: 39 ms Size: 344 B

En este caso el status es 302, que quiere decir Found, encontrado

```
1 {
2   "id": 5,
3   "nombre": "Antonio",
4   "apellido1": "Carretero",
5   "apellido2": "Ortega",
6   "comision": 0.12
7 }
```

El resultado de esta búsqueda, nos muestra el comercial que tiene asignado el id 5, ya que le hemos pasado por el endpoint el 5.

Ahora vamos a realizar un alta de un nuevo comercial, para el cual usaremos el método http POST y el endpoint sería así, <http://localhost:8083/comercial/alta> tendríamos que mandar el nuevo comercial en JSON.

POST <http://localhost:8083/comercial/alta>

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

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

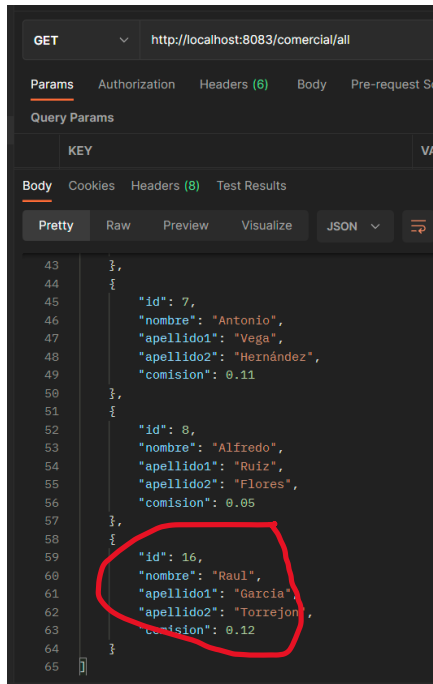
```
1 {
2   "nombre": "Raul",
3   "apellido1": "Garcia",
4   "apellido2": "Torrejon",
5   "comision": 0.12
6 }
```

Podemos observar que el status es 201 Created, que nos confirma que ha sido creado

Status: 201 Created Time: 58 ms Size: 343 B

```
1 {
2   "id": 16,
3   "nombre": "Raul",
4   "apellido1": "Garcia",
5   "apellido2": "Torrejon",
6   "comision": 0.12
7 }
```

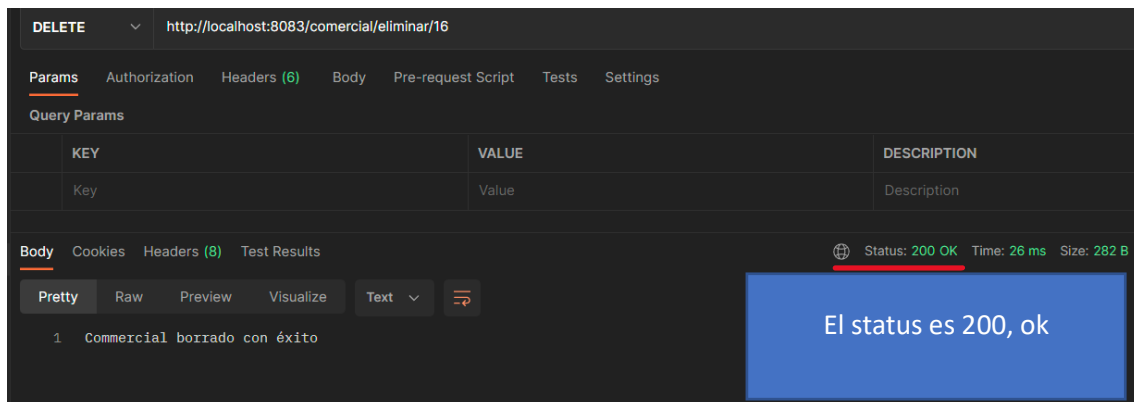
También recibimos en la respuesta, el comercial creado, que en este caso tendría el id 16, ahora vamos a comprobar otra vez con el método GET a la llamada del endpoint que nos trae todos los comerciales y comprobamos que si que se ha creado.



```
43  },
44  },
45  {
46    "id": 7,
47    "nombre": "Antonio",
48    "apellido1": "Vega",
49    "apellido2": "Hernández",
50    "comision": 0.11
51  },
52  {
53    "id": 8,
54    "nombre": "Alfredo",
55    "apellido1": "Ruiz",
56    "apellido2": "Flores",
57    "comision": 0.05
58  },
59  {
60    "id": 16,
61    "nombre": "Raul",
62    "apellido1": "Garcia",
63    "apellido2": "Torrejon",
64    "comision": 0.12
65  }
66 }
```

Podemos comprobar, que sí que se ha persistido en la bbdd y hemos creado un nuevo comercial, con id 16, llamado Raul y sus datos.

Ahora vamos a usar el método DELETE para eliminar un comercial, en este caso vamos a eliminar el comercial que hemos creado anteriormente, con id 16, este sería el endpoint <http://localhost:8083/comercial/eliminar/16>



DELETE http://localhost:8083/comercial/eliminar/16

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (8) Test Results

Status: 200 OK Time: 26 ms Size: 282 B

1 Commercial borrado con éxito

El status es 200, ok

En este caso, vemos como poniendo un id que, si existe, aparte de eliminarlo, nos manda un mensaje que nos confirma que ha sido borrado con éxito

DELETE `http://localhost:8083/comercial/eliminar/16`

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (8) Test Results

Status: 404 Not Found Time: 8 ms Size: 283 B

Pretty Raw Preview Visualize Text

1 Comercial no encontrado

En este caso el status no es 200, si no, 404 Not found

Por el contrario, si volvemos a realizar otra llamada con el mismo ID, que en este caso ya ha sido eliminado, nos enviará el mensaje de Comercial no encontrado

De nuevo haciendo uso del método GET vamos a realizar una consulta para devlver la lista de los comerciales que han atendido pedidos del cliente que coincida con el id pasado, el endpoint sería el siguiente, <http://localhost:8083/comercial/bycliente/1>

GET `http://localhost:8083/comercial/bycliente/1`

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (8) Test Results

Status: 302 Found Time: 17 ms Size: 344 B

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 5,
3   "nombre": "Antonio",
4   "apellido1": "Carretero",
5   "apellido2": "Ortega",
6   "comision": 0.12
7 }
```

Status 302, Found, que nos confirma que ha sido econtrado

Hemos pedido que nos devuelva el comercial que ha atendido al cliente con id 1, en este caso, nos devuelve el comercial con ID 5.

Sin embargo, si buscamos el comercial que ha atendido al cliente con ID 100 que no existe, nos devolvería

GET `http://localhost:8083/comercial/bycliente/100`

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (8) Test Results

Status: 404 Not Found Time: 13 ms Size: 283 B

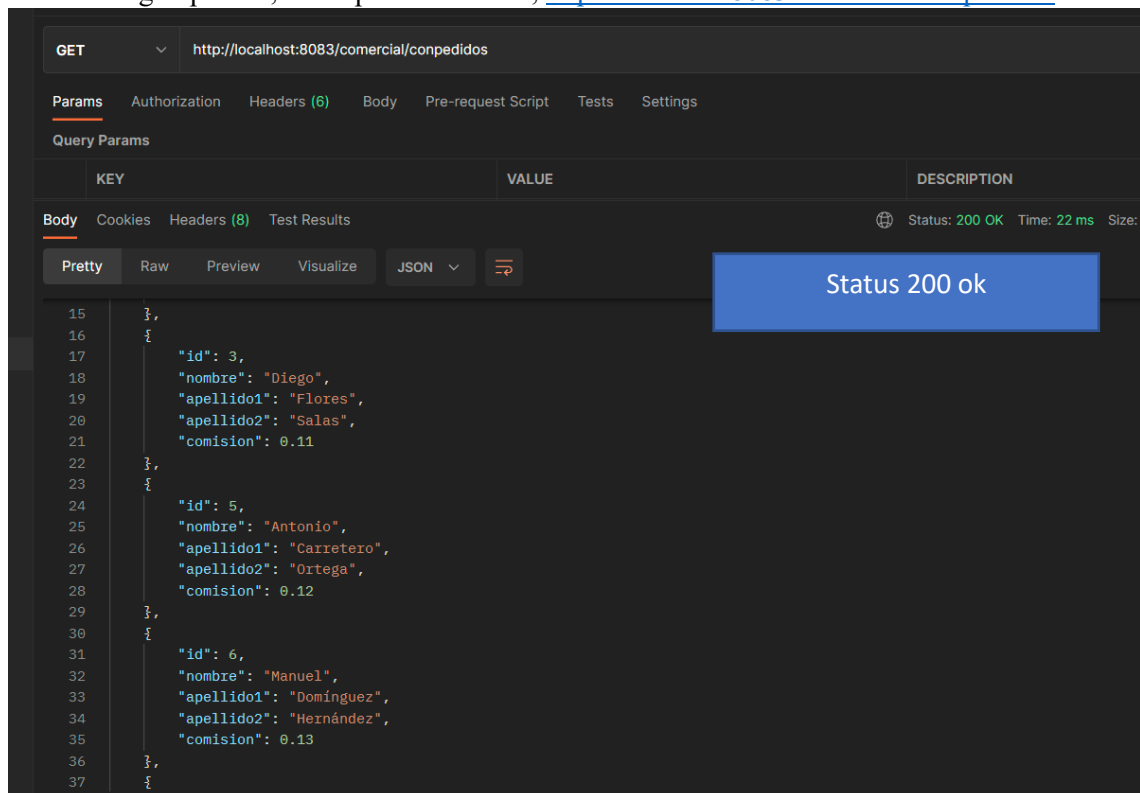
Pretty Raw Preview Visualize Text

1 Comercial no encontrado

Mensaje de no encontrado y status 404 not found



Vamos a volver a usar el método GET para que nos devuelva una lista de comerciales que han atendido algún pedido, el endpoint sería este, <http://localhost:8083/comercial/conpedidos>



GET <http://localhost:8083/comercial/conpedidos>

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
-----	-------	-------------

Body Cookies Headers (8) Test Results Status: 200 OK Time: 22 ms Size: 100 B

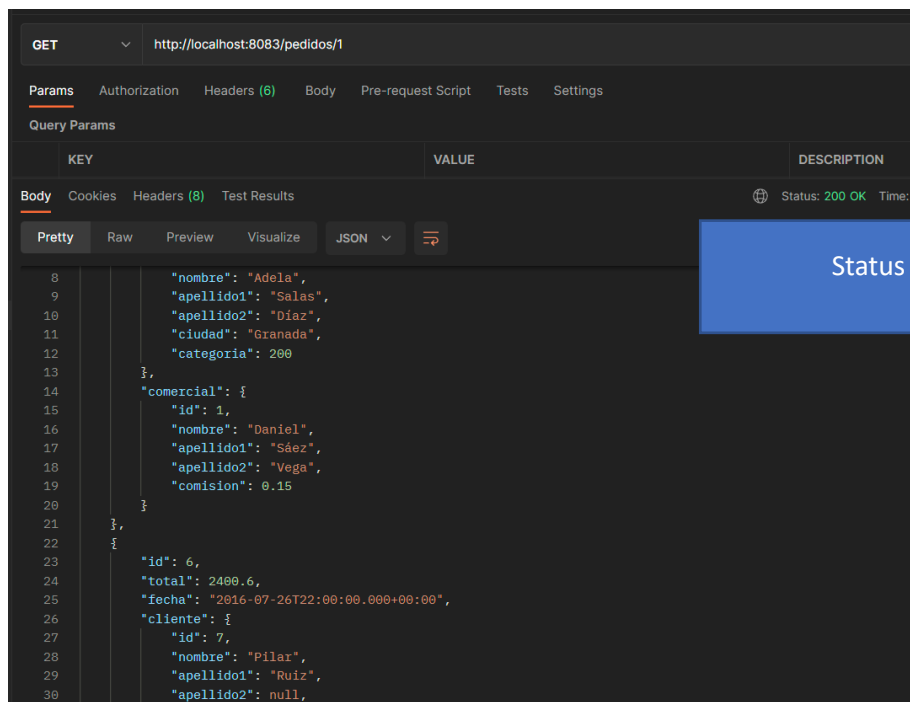
Pretty Raw Preview Visualize JSON ↕

```
15  {
16    {
17      "id": 3,
18      "nombre": "Diego",
19      "apellido1": "Flores",
20      "apellido2": "Salas",
21      "comision": 0.11
22    },
23    {
24      "id": 5,
25      "nombre": "Antonio",
26      "apellido1": "Carretero",
27      "apellido2": "Ortega",
28      "comision": 0.12
29    },
30    {
31      "id": 6,
32      "nombre": "Manuel",
33      "apellido1": "Dominguez",
34      "apellido2": "Hernández",
35      "comision": 0.13
36    },
37  }
```

Status 200 ok

Podemos ver que el comercial con id 4, no tiene ningún pedido, por lo que no se muestra.

Por último vamos a usar de nuevo el método GET para que nos devuelva una lista de pedidos gestionados por el comercial que coincida con el id pasado por el endpoint, el endpoint sería el siguiente, <http://localhost:8083/pedidos/1>



GET <http://localhost:8083/pedidos/1>

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
-----	-------	-------------

Body Cookies Headers (8) Test Results Status: 200 OK Time: 4 ms Size: 100 B

Pretty Raw Preview Visualize JSON ↕

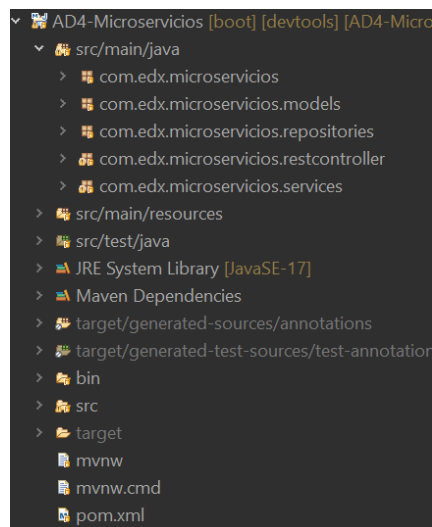
```
8    "nombre": "Adela",
9    "apellido1": "Salas",
10   "apellido2": "Díaz",
11   "ciudad": "Granada",
12   "categoria": 200
13 },
14 "comercial": {
15   "id": 1,
16   "nombre": "Daniel",
17   "apellido1": "Sáez",
18   "apellido2": "Vega",
19   "comision": 0.15
20 },
21 },
22 {
23   "id": 6,
24   "total": 2400.6,
25   "fecha": "2016-07-26T22:00:00.000+00:00",
26   "cliente": {
27     "id": 7,
28     "nombre": "Pilar",
29     "apellido1": "Ruiz",
30     "apellido2": null,
```

Status 200 ok

En este caso nos muestra toda la información del pedido, nos da el id, el total, la fecha, el cliente que es con su id, nombre... Demasiada información, si tuviéramos el caso en el que queremos mostrar información específica, como por ejemplo del cliente solo el nombre, y del comercial el id, mostrar también el precio total, deberíamos montarnos una clase intermedia, en un paquete DTO, y ahí crear una nueva clase que sólo tenga como atributos, los que queramos enseñar y a la hora de llamar al endpoint, en vez de buscar el objeto con toda la información, llamaríamos al DTO, ojo, este DTO simplemente es para consultas, no debemos hacer un alta ya que nos faltaría información que es relevante para persistir en la bbdd.

## Código:

Primero vamos a mostrar cómo está distribuido nuestra aplicación.



Podemos observar que tenemos 5 paquetes, el primero es el que se nos crea al crear nuestro spring starter con Maven.

A continuación, tenemos uno que es models o entities, donde tendremos nuestros java beans para persistir con bbdd.

El paquete repositories, que será donde tengamos alojados nuestro repositorios que extienda de JpaRepository y donde añadiremos algún método abstracto con consultas SQL para hacer búsquedas personalizadas.

El paquete restcontroller, que será donde tengamos todos nuestros endpoint para hacer las diferentes llamadas.

Y por último nuestro paquete service, que sería el mismo que el paquete dao que usamos para las aplicaciones web full stack java, aquí crearemos nuestras interfaces intermedias para poder elegir los métodos que vamos a usar y así no visualizar todos los métodos que nos proporciona JpaRepository, que también los podemos usar.

Vamos a mostrar todos los javabeans que tienen anotaciones JPA para persistir las diferentes tablas en la bbdd,

Cliente	Comercial
<pre> 1 package com.edx.microservicios.models; 2 3 import javax.persistence.Column; 4 5 @Entity 6 @Table(name = "clientes") 7 public class Cliente { 8 9     @Id 10    @GeneratedValue(strategy = GenerationType.IDENTITY) 11    @Column(name = "id_cliente") 12    private int id; 13 14    @Column(nullable = false) 15    private String nombre; 16 17    @Column(nullable = false) 18    private String apellido1; 19 20    private String apellido2; 21 22    private String ciudad; 23 24    private int categoria; 25 26    public Cliente() { 27    } 28 </pre>	<pre> 1 package com.edx.microservicios.models; 2 3 import javax.persistence.Column; 4 5 @Entity 6 @Table(name = "comerciales") 7 public class Comercial { 8 9     @Id 10    @GeneratedValue(strategy = GenerationType.IDENTITY) 11    @Column(name = "id_comercial") 12    private int id; 13 14    @Column(nullable = false) 15    private String nombre; 16 17    @Column(nullable = false) 18    private String apellido1; 19 20    private String apellido2; 21 22    private double comision; 23 24    public Comercial() { 25    } 26 </pre>

Pedido
<pre> 1 package com.edx.microservicios.models; 2 3 import java.util.Date; 4 5 @Entity 6 @Table(name = "pedidos") 7 public class Pedido { 8 9     @Id 10    @GeneratedValue(strategy = GenerationType.IDENTITY) 11    @Column(name = "id_pedido") 12    private int id; 13 14    @Column(nullable = false) 15    private double total; 16 17    private Date fecha; 18 19    @ManyToOne 20    @JoinColumn(name = "id_cliente", nullable = false) 21    private Cliente cliente; 22 23    @ManyToOne 24    @JoinColumn(name = "id_comercial", nullable = false) 25    private Comercial comercial; 26 27    public Pedido() { 28    } 29 </pre>

Vamos a mostrar el paquete repositories, como hemos comentado anteriormente, aquí tendremos una interface que extienda de JpaRepository y si necesitamos hacer alguna consulta específica deberemos de crear nuestros métodos para ello, ahora vemos algún ejemplo.

Cliente	Comercial
<pre> 1 package com.edx.microservicios.repositories; 2 3 import org.springframework.data.jpa.repository.JpaRepository; 4 5 public interface ClienteRepository extends JpaRepository&lt;Cliente, Integer&gt;{ 6 7 } 8 9 10 </pre>	<pre> 1 import java.util.List; 2 3 public interface ComercialRepository extends JpaRepository&lt;Comercial, Integer&gt;{ 4 5     @Query("Select distinct c from Comercial c, Pedido p where p.comercial.id = c and p.cliente.id = ?1") 6     public Optional&lt;Comercial&gt; findByIdCliente(int id); 7 8     @Query("Select distinct c from Comercial c, Pedido p where p.comercial.id = c") 9     public List&lt;Comercial&gt; buscarConPedido(); 10 </pre>

Pedido
<pre> 1 import java.util.List; 2 3 public interface PedidoRepository extends JpaRepository&lt;Pedido, Integer&gt;{ 4 5     @Query("Select p from Pedido p where p.comercial.id = ?1") 6     public List&lt;Pedido&gt; buscarComercialConPedido(int id); 7 </pre>

Primero vamos a ver el paquete services y luego por último mostraremos el restcontroller, vamos a ello (en este caso no hemos implementado los servicios para cliente, ya que no vamos a hacer ninguna consulta, de momento, podría añadirse en un futuro si así se requiere)

Comercial	Pedido
<pre> 1 2 3*import java.util.List; 4 5 6 7 8 9 public interface ComercialService { 10 11     Comercial altaComercial(Comercial comercial); 12     boolean eliminarComercial(int id); 13     Optional&lt;Comercial&gt; findById(int id); 14     List&lt;Comercial&gt; findAll(); 15     Optional&lt;Comercial&gt; buscarComercialPorCliente(int idCliente); 16     List&lt;Comercial&gt; buscarConPedido(); 17 18 19 20 21 } 22 23 </pre>	<pre> 1 2 3*import java.util.List; 4 5 6 7 8 9 public interface PedidoService { 10 11     List&lt;Pedido&gt; pedidoPorComercial(int idComercial); 12 13 14 } 15 </pre>

Estas interfaces son como nuestro “contrato” a la hora de implementarlo en las clases que vamos a ver a continuación, nos marcan los métodos que vamos a tener que implementar, como mínimo, los que hemos definido en ella.

Veamos las clases que implementan estas interfaces:

Comercial	Pedido
<pre> 1 package com.eux.microservicios.services; 2 3*import java.util.List; 4 5 6 @Service 7 public class ComercialServiceImpl8 implements ComercialService{ 8 9     //Para poder hacer uso del repositorio le instanciamos con la anotación @Autowired 10    @Autowired 11    private ComercialRepository crep; 12 13 14    @Override 15    // Nuevo comercial 16    public Comercial altaComercial(Comercial comercial) { 17        return crep.save(comercial); 18    } 19 20    @Override 21    // Eliminar comercial 22    public boolean eliminarComercial(int id) { 23        try { 24            crep.deleteById(id); 25            return true; 26        } catch (Exception err) { 27            return false; 28        } 29    } 30 31    @Override 32    // Buscar comercial por id 33    public Optional&lt;Comercial&gt; findById(int id) { 34        return crep.findById(id); 35    } 36 37    @Override 38    // Mostrar todos 39    public List&lt;Comercial&gt; findAll() { 40        return crep.findAll(); 41    } 42 43    @Override 44    public Optional&lt;Comercial&gt; buscarComercialPorCliente(int idCliente) { 45        return crep.findAllByCliente(idCliente); 46    } 47 48    @Override 49    public List&lt;Comercial&gt; buscarConPedido() { 50        return crep.buscarConPedido(); 51    } 52 </pre>	<pre> 1 2 3*import java.util.List; 4 5 6 7 8 9 @Service 10 public class PedidoServiceImpl8 implements PedidoService{ 11 12 13    @Autowired 14    private PedidoRepository pRep; 15 16 17    @Override 18    public List&lt;Pedido&gt; pedidoPorComercial(int idComercial) { 19        return pRep.buscarComercialConPedido(idComercial); 20    } 21 22 23 } 24 </pre>

Como se puede apreciar, se han implementado todos los métodos creados en las interfaces que implementa cada una de las clases.

Vamos a ver por último el paquete restcontroller y cada uno de ellos

## Comercial

```
1 package com.edx.microservicios.restcontroller;
2
3 import java.util.List;
4
5 //La anotación @CrossOrigins(origins="*"), es para permitir accesos desde aplicaciones cliente web
6 @CrossOrigin(origins = "*")
7 @RestController
8 @RequestMapping("/comercial")
9 public class ComercialRestController {
10
11     @Autowired
12     private ComercialService cserv;
13
14     //Mostrar todos los comerciales
15     @GetMapping("/all")
16     public List<Comercial> mostrarTodos(){
17         return cserv.findAll();
18     }
19
20     //Mostramos el comercial buscado por Id
21     @GetMapping("/uno/{id}")
22     public ResponseEntity<?> buscarPorId(@PathVariable("id") int id) {
23         //si el comercial buscado por el id pasado por parametro no existe
24         if (!cserv.findById(id).isPresent()) {
25             //devolvemos el estado not found (404) y el mensaje que queremos añadir
26             return ResponseEntity.status(HttpStatus.NOT_FOUND).body("Comercial no encontrado");
27         }
28         //devolvemos el estado found (302) y el objeto que buscamos en el body
29         return ResponseEntity.status(HttpStatus.FOUND).body(cserv.findById(id));
30     }
31
32     @PostMapping("/alta")
33     public ResponseEntity<?> altaComercial(@RequestBody Comercial comercial) {
34         //Devolvemos el estado created 201 y le metemos en el cuerpo el comercial creado
35         return ResponseEntity.status(HttpStatus.CREATED).body(cserv.altaComercial(comercial));
36     }
37
38     @DeleteMapping("/eliminar/{id}")
39     public ResponseEntity<?> borrarComercial(@PathVariable("id") int id) {
40         if (!cserv.findById(id).isPresent()) {
41             //Si llegamos aquí mandamos not found (404) y el mensaje en el body
42             return ResponseEntity.status(HttpStatus.NOT_FOUND).body("Comercial no encontrado");
43         }
44         //Eliminamos comercial
45         cserv.eliminarComercial(id);
46         //En este caso devolvemos el status ok (200) y un mensaje en el body
47         return ResponseEntity.status(HttpStatus.OK).body("Comercial borrado con éxito");
48     }
49
50     @GetMapping("/bycliente/{id}")
51     public ResponseEntity<?> buscarPorCliente(@PathVariable("id") int idCliente){
52         if(!cserv.findById(idCliente).isPresent()) {
53             //Si llegamos aquí mandamos not found (404) y el mensaje en el body
54             return ResponseEntity.status(HttpStatus.NOT_FOUND).body("Comercial no encontrado");
55         }
56         //En este caso devolvemos el status found (302) y un mensaje en el body
57         return ResponseEntity.status(HttpStatus.FOUND).body(cserv.buscarComercialPorCliente(idCliente));
58     }
59
60     @GetMapping("/conpedidos")
61     public List<Comercial> buscarConPedidos() {
62         return cserv.buscarConPedido();
63     }
64 }
```

## Pedido

```
13 import java.util.List;
14
15 //La anotación @CrossOrigins(origins="*"), es para permitir accesos desde aplicaciones cliente web
16 @CrossOrigin(origins = "*")
17 @RestController
18 @RequestMapping("/pedidos")
19 public class PedidoRestController {
20
21     @Autowired
22     private PedidoService pserv;
23
24     @GetMapping("/{id}")
25     public List<Pedido> buscarPorComercial(@PathVariable("id") int id){
26         return pserv.pedidoPorComercial(id);
27     }
28
29 }
30
```

## Conclusiones

Hemos montado un microservicio que escucha peticiones HTTP, mediante el cual podemos persistir en la bbdd, y podemos dar de alta comerciales, buscar por diferentes medios, ya sea por id del comercial ,por los pedidos que tiene... y eliminar, para que fuera un CRUD entero nos faltaría un método para poder actualizar las entidades.