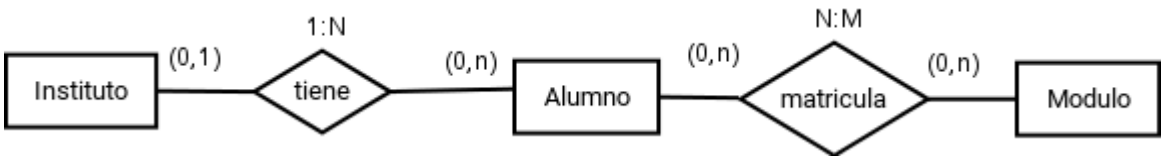
En esta breve memoria voy a hacer una descripción del problema resuelto, desde su idea hasta su conclusión con un API Rest que proporciona servicios principales CRUD y algunos adicionales sobre una de las tablas principales de la base de datos.

Queremos registrar en nuestra base de datos una serie de institutos, que tienen alumnos perteneciendo estos alumnos pertenecen a un único instituto. Por otra parte, tenemos una serie de módulos que pueden cursar los alumnos, por lo que un alumno puede estar matriculado en muchos módulos y un módulo puede tener muchos alumnos matriculados.

El Modelo entidad relación de estas tablas sería el siguiente



El modelo relacional resultante es el siguiente

```sql
DROP DATABASE IF EXISTS institutosMatricula;
CREATE DATABASE institutosMatricula CHARACTER SET utf8mb4;
USE institutosMatricula;

CREATE TABLE IF NOT EXISTS TInstituto (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(255) not null,
    direccion VARCHAR(255) not null,
    localidad VARCHAR(100) not null,
    telefono VARCHAR(15) not null);

CREATE TABLE IF NOT EXISTS TAlumno (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(255) not null,
    fecha_nacimiento DATE,
    direccion VARCHAR(255),
    correo VARCHAR(255),
    telefono VARCHAR(15),
    instituto_id INT,
    constraint fk_alumno_instituto FOREIGN KEY (instituto_id)
        REFERENCES TInstituto(id) On delete restrict On Update Cascade
);
```

```sql
CREATE TABLE IF NOT EXISTS TModulo (
    codigo_modulo INT AUTO_INCREMENT PRIMARY KEY,
    nombre_modulo VARCHAR(255) NOT NULL,
    ciclo_formativo VARCHAR(255) NOT NULL
);

CREATE TABLE IF NOT EXISTS TMatricula (
    alumno_id INT,
    modulo_codigo INT,
    PRIMARY KEY (alumno_id, modulo_codigo),
    FOREIGN KEY (alumno_id) REFERENCES TAlumno(id) ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY (modulo_codigo) REFERENCES TModulo(codigo_modulo) ON DELETE RESTRICT ON UPDATE CASCADE
);
```

Además nuestro Script incorpora algunas inserciones para poder hacer pruebas.

```sql
INSERT INTO TInstituto (nombre, direccion, localidad, telefono) VALUES
('IES Vicente Medina', 'C/. Vaso Ibérico de los Guerreros, S/N', 'ARCHENA', '968670157'),
('IES Los Albares', 'C/ Vereda de Morcillo S/N', 'CIEZA', '968773077'),
('IES Valle del Segura', 'Av. Río Segura, nº 10', 'BLANCA', '968459348');

INSERT INTO TAlumno (nombre, fecha_nacimiento, direccion, correo, telefono, instituto_id) VALUES
('Juan Pérez', '2000-01-15', 'Calle 123, Ciudad A', 'juan@example.com', '123456789', 1),
('Ana García', '1999-05-22', 'Avenida XYZ, Ciudad B', 'ana@example.com', '987654321', 2),
('Pedro López', '1998-11-10', 'Calle Principal, Ciudad A', 'pedro@example.com', '555555555', 1),
('María Rodríguez', '2001-03-05', 'Calle Secundaria, Ciudad C', 'maria@example.com', '777777777', 3),
('Carlos Sánchez', '1997-08-18', 'Avenida Central, Ciudad B', 'carlos@example.com', '999999999', 2);

INSERT INTO TModulo (nombre_modulo, ciclo_formativo) VALUES
('Acceso a Datos', 'Desarrollo de Aplicaciones Multiplataforma'),
('Seguridad y Alta Disponibilidad', 'Administración de Sistemas Informáticos en Red'),
('Programación', 'Desarrollo de Aplicaciones Multiplataforma'),
('Redes de Computadoras', 'Administración de Sistemas Informáticos en Red');

INSERT INTO TMatricula (alumno_id, modulo_codigo) VALUES
(1, 1),(1, 2),(2, 2),(3, 3),(4, 4);
```

Este Script está adjunto con el nombre institutosMatricula.sql

El siguiente ha sido crear nuestro proyecto con Spring Boot (el proyecto va adjunto también a la tarea) para crear un API Rest a partir del mismo. Para ello hemos usado Initializr y hemos añadido la librería Lombok que nos permite generar Setter, Getter y algunas cosas más con mínimo código (con anotaciones) y DevTools que nos va a permitir hacer reinicio rápido. Por supuesto, también hemos añadido la librería para MySQL, Spring Data JPA y Spring Web que la necesitamos ya que el fin de nuestra aplicación es crear el API Rest.

Para añadir las clases vamos a seguir esta recomendación de estructura de paquetes recomendado para Spring Boot https://gustavopeiretti.com/estructura-de-paquetes-spring-boot/

Vamos a generar las entidades del modelo. Aunque tenemos 4 tablas vamos a tener 3 entidades únicamente ya que la tabla TMatricula se va a resolver con anotaciones @ManyToMany en las entidades correspondientes.

Las relaciones que hemos definido son

- En Alumno tenemos una relación @ManyToMany con Modulo y una relación @ManyToOne con Instituto (muchos alumnos tienen el mismo instituto)
- En Instituto tenemos un @OneToMany a Alumno (un Instituto tiene muchos alumnos).
- En Módulo tenemos un @ManyToMany con Alumno.

Deberemos definir un repositorio por entidad, ellos van a extender JpaRepository por lo que incorporan automáticamente el CRUD para cada entidad. En el caso de IInstitutoRepository añadiré también un par de métodos más para obtener los institutos por localidad (así puedo probar la funcionalidad que Spring Data Jpa nos proporciona de hacer consultas únicamente con el nombre del método) y obtener los institutos cuyo teléfono comiencen por un número (para probar las JPQL)

```
public List<Instituto> findByLocalidad(String localidad);

@Query ("FROM Instituto i WHERE LOWER(i.telefono) LIKE :numero%")
List<Instituto> listarPorPrefijoTelefono(@Param("numero") String numero);
```

Creamos una interfaz con los servicios que vamos a implementar con Instituto y que implementaremos utilizando los métodos que nos proporciona Spring Data (y estos dos que hemos indicado). Estos son los que hemos definido.

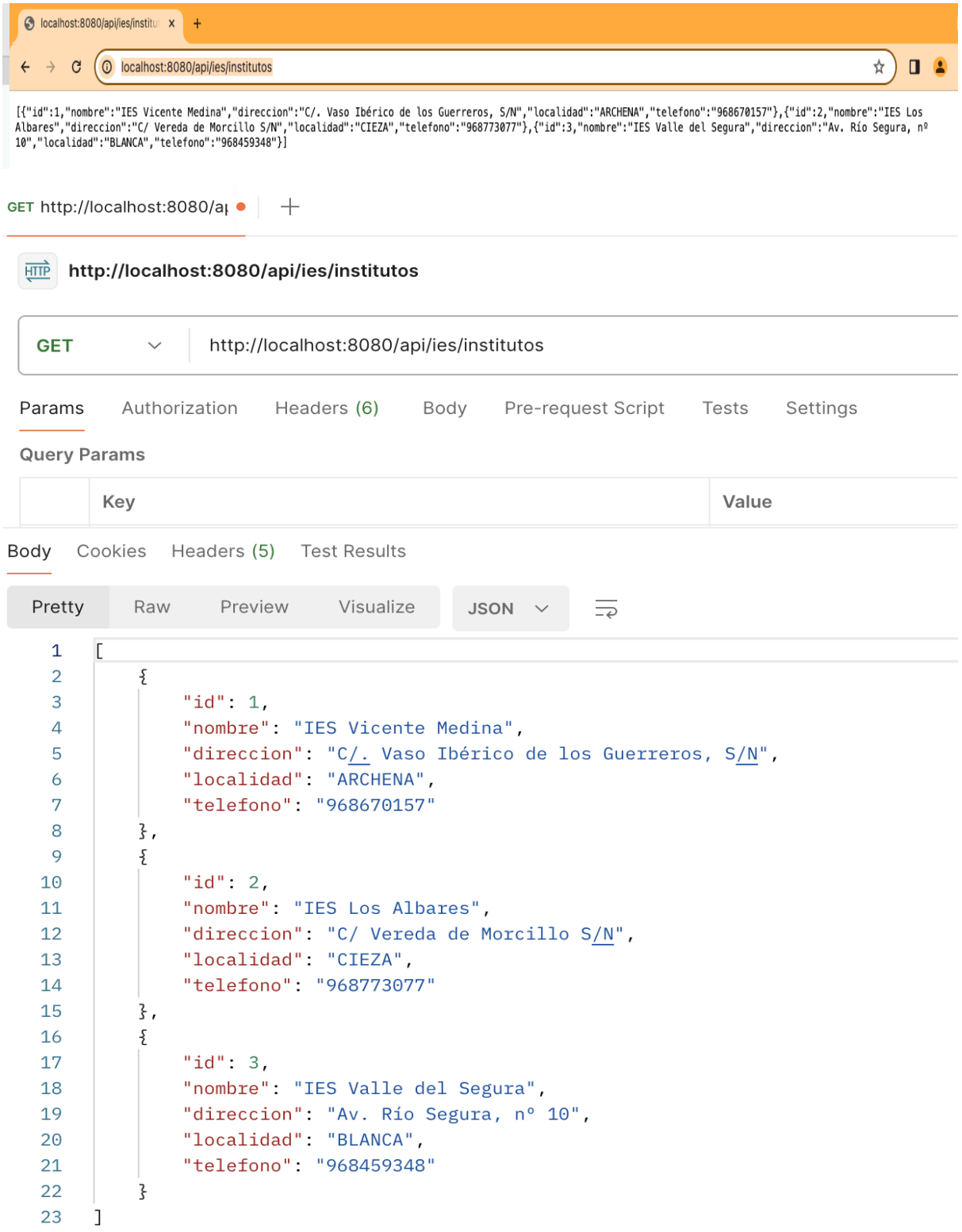
```
Instituto registrar (Instituto instituto);
Instituto modificar (Instituto instituto);
ArrayList<Instituto> listar();
Instituto obtener (Integer id);
void eliminar (Integer id);
ArrayList<Instituto> listarPorLocalidad(String localidad);
ArrayList<Instituto> listarPorPrefijo(String prefijo);
```

Por último, aprovechando esta estructura y utilizando las anotaciones de Spring Boot adecuadas (y algunas Jackson, Lombok o Hibernate) hemos definido nuestra clase controller donde vamos a tener los endpoints que proporcionamos a los usuarios y que se pueden acceder a ellos desde un navegador (métodos get), postman o incluso si se quisiera desde otra aplicación JAVA Cliente.

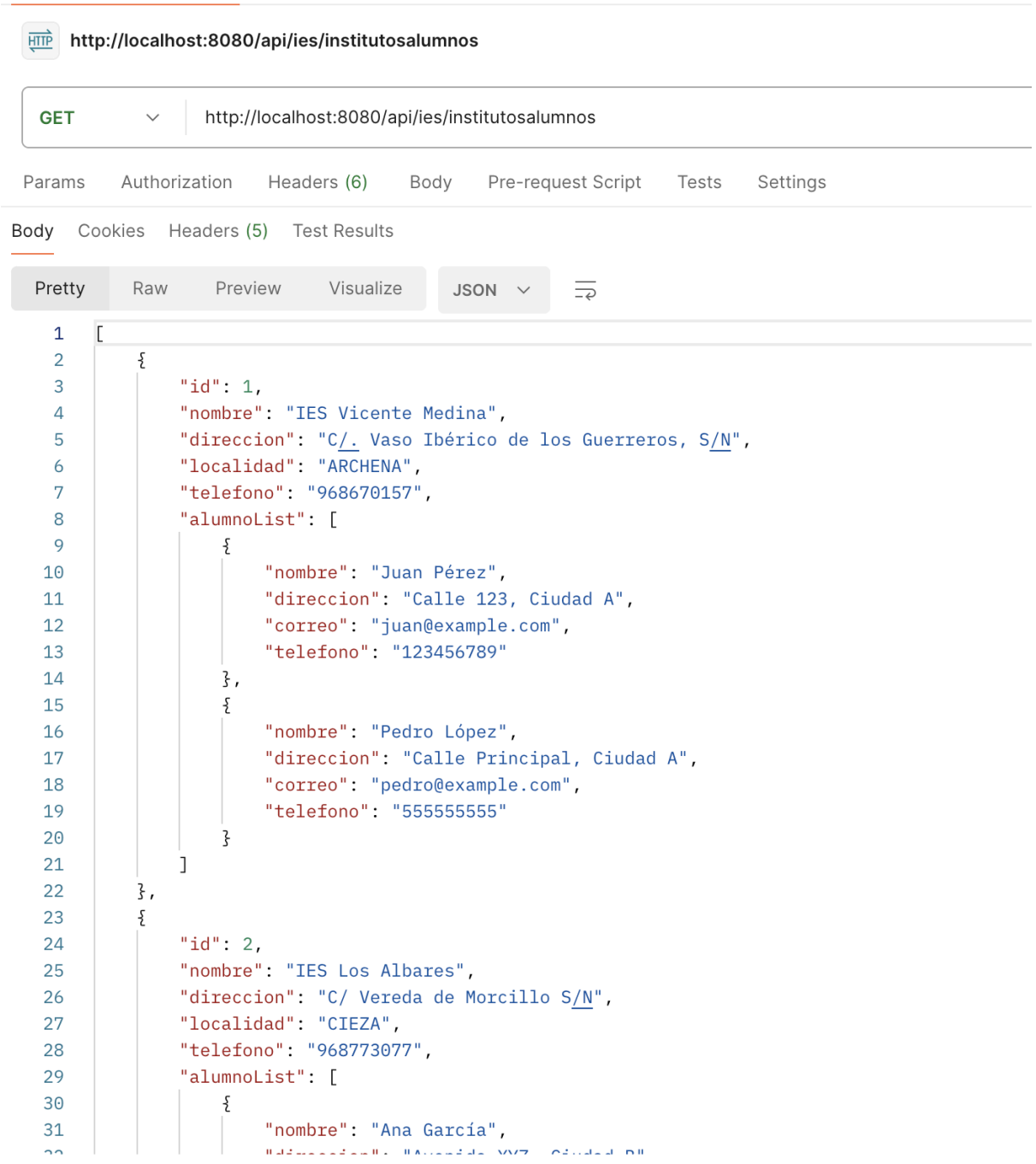
Aunque alguna prueba la mostraré en navegador la mayoría emplearé Postman ya que te ofrece una mejor visualización de los resultados.

Endpoints

- Obtener todos los institutos → Ruta: http://localhost:8080/api/ies/institutos

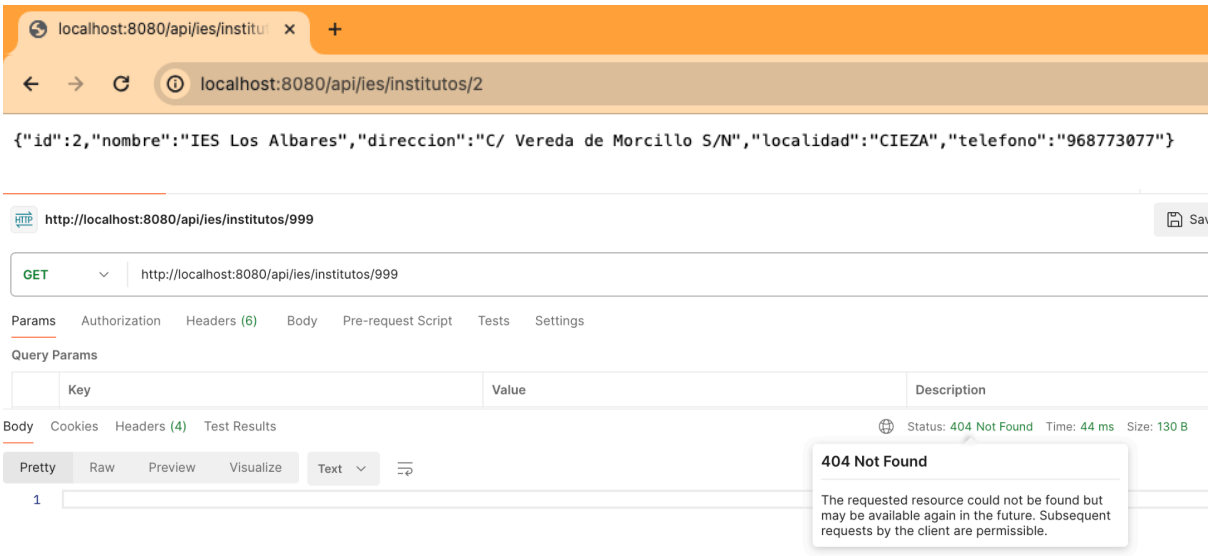


- Obtener los institutos con sus alumnos. Para hacer esta parte hemos tenido que crear unas clases especiales DTO (Data Transfer Object), replicando la clase Instituto y Alumno con los datos que nos interesan y que permite mostrar estos datos sin tener un bucle infinito. Ruta → http://localhost:8080/api/ies/institutosalumnos



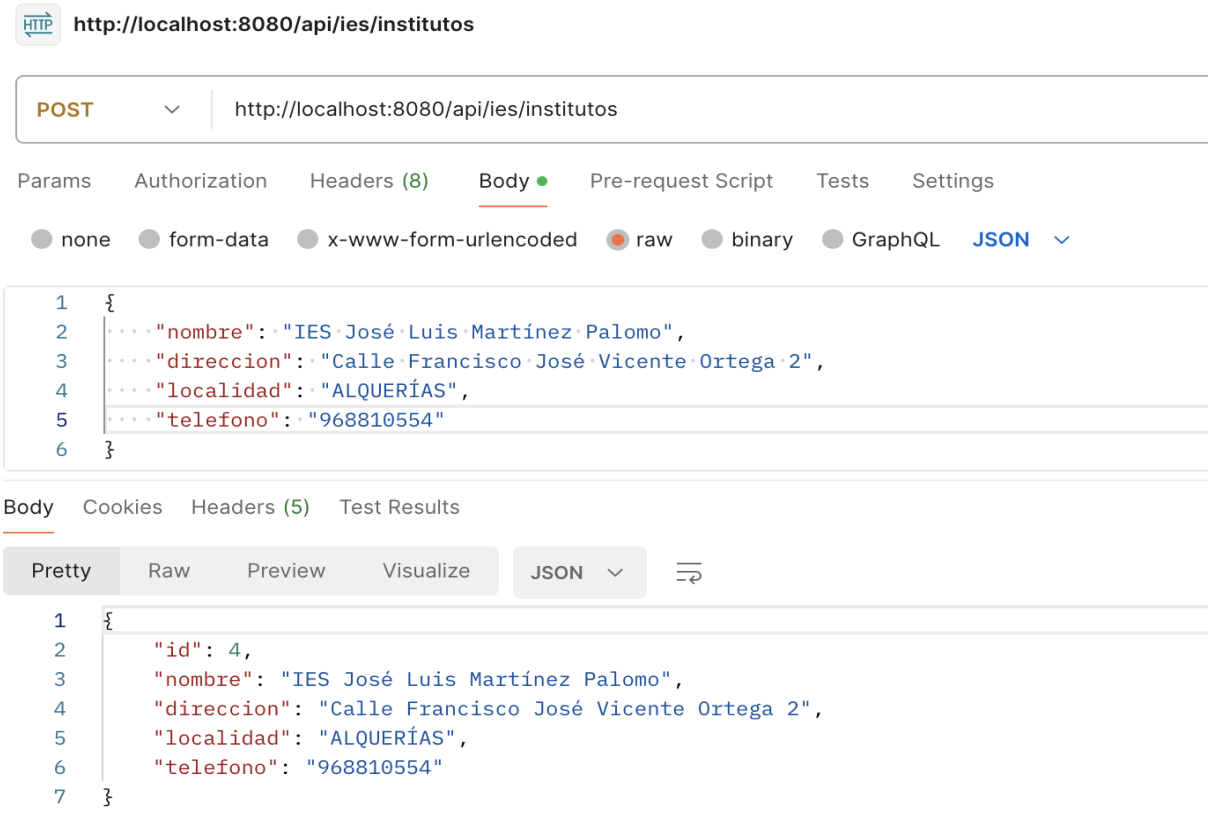

```
[
    {
        "id": 1,
        "nombre": "IES Vicente Medina",
        "direccion": "C/. Vaso Ibérico de los Guerreros, S/N",
        "localidad": "ARCHENA",
        "telefono": "968670157",
        "alumnoList": [
            {
                "nombre": "Juan Pérez",
                "direccion": "Calle 123, Ciudad A",
                "correo": "juan@example.com",
                "telefono": "123456789"
            },
            {
                "nombre": "Pedro López",
                "direccion": "Calle Principal, Ciudad A",
                "correo": "pedro@example.com",
                "telefono": "555555555"
            }
        ]
    },
    {
        "id": 2,
        "nombre": "IES Los Albares",
        "direccion": "C/ Vereda de Morcillo S/N",
        "localidad": "CIEZA",
        "telefono": "968773077",
        "alumnoList": [
            {
                "nombre": "Ana García",
```

- Obtener un instituto por su id. Ruta → http://localhost:8080/api/ies/institutos/{id}



Los siguientes métodos ya no se podrían probar desde el navegador ya que reciben los parámetros en el body, por lo que ya no pondré la ruta del endpoint ya que no tendría sentido pulsar en ella (se puede ver en postman)

- Insertar un instituto. Los datos los recibe en el Body con un JSON.



```
57 • Select * From TInstituto;
58
```

00% 26:57

Result Grid Filter Rows: Search Edit: Export/Import:

id	nombre	direccion	localidad	telefono
1	IES Vicente Medina	C/. Vaso Ibérico de los Guerreros, S/N	ARCHENA	968670157
2	IES Los Albares	C/ Vereda de Morcillo S/N	CIEZA	968773077
3	IES Valle del Segura	Av. Río Segura, nº 10	BLANCA	968459348
4	IES José Luis Martínez Palomo	Calle Francisco José Vicente Ortega 2	ALQUERÍAS	968810554
NULL	NULL	NULL	NULL	NULL

- Actualizar un instituto. En el endpoint que hemos definido recibe el id del instituto en la ruta y el resto de los datos en un JSON. Actualiza los datos para el id del instituto que recibe. En el siguiente ejemplo vamos a cambiar la localidad para poner MURCIA en el instituto 4.

PUT http://localhost:8080/api/ies/institutos/4

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

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

```
{
    "nombre": "IES José Luis Martínez Palomo",
    "direccion": "Calle Francisco José Vicente Ortega 2",
    "localidad": "MURCIA",
    "telefono": "968810554"
}
```

Body Cookies Headers (5) Test Results Status: 201 Created Tim

Pretty Raw Preview Visualize JSON

```
{
    "id": 4,
    "nombre": "IES José Luis Martínez Palomo",
    "direccion": "Calle Francisco José Vicente Ortega 2",
    "localidad": "MURCIA",
    "telefono": "968810554"
}
```

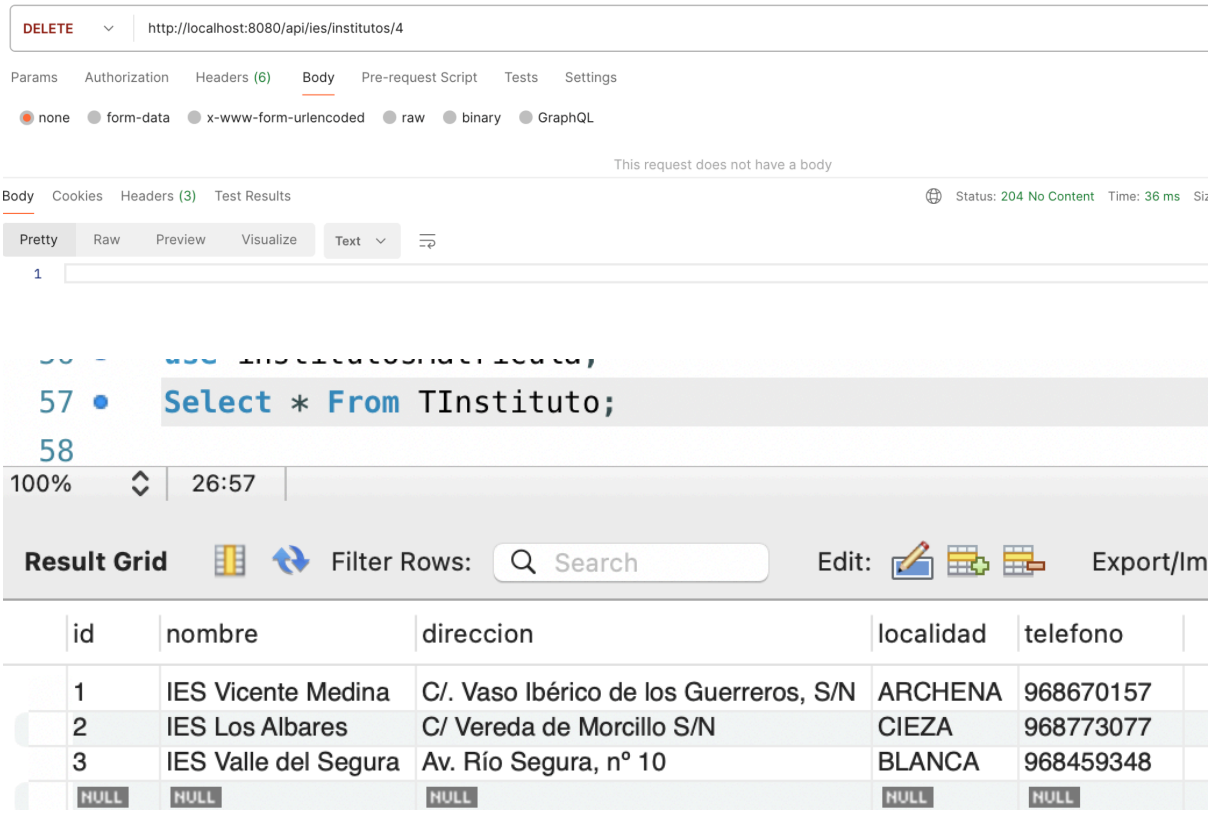
```
57 • Select * From TInstituto;
58
```

100% 26:57

Result Grid Filter Rows: Search Edit: Export/Import:

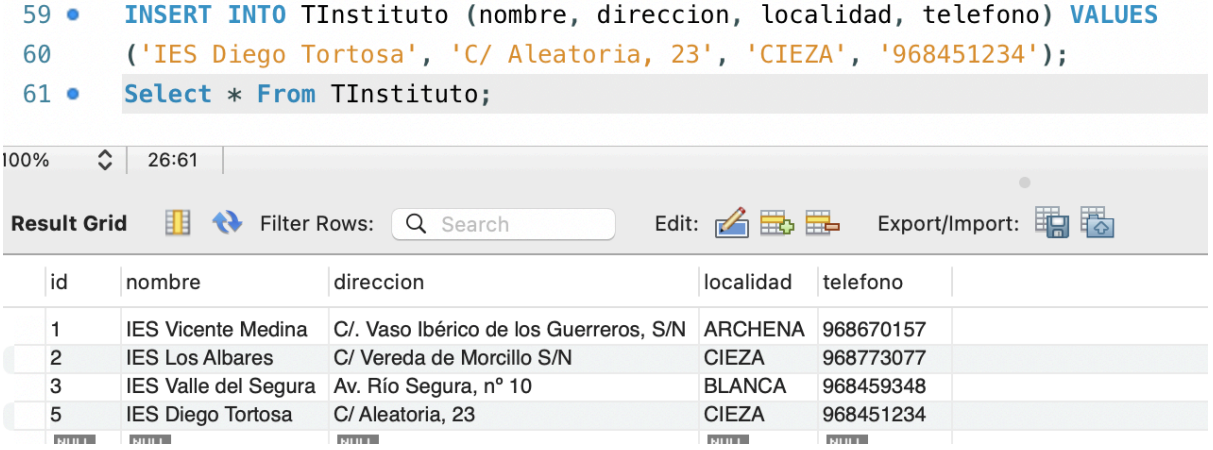
id	nombre	direccion	localidad	telefono
1	IES Vicente Medina	C/. Vaso Ibérico de los Guerreros, S/N	ARCHENA	968670157
2	IES Los Albares	C/ Vereda de Morcillo S/N	CIEZA	968773077
3	IES Valle del Segura	Av. Río Segura, nº 10	BLANCA	968459348
4	IES José Luis Martínez Palomo	Calle Francisco José Vicente Ortega 2	MURCIA	968810554
NULL	NULL	NULL	NULL	NULL

- Eliminar un instituto se podría utilizar desde el navegador ya que el único parámetro que recibe un instituto pero no es un método GET (es un PUT) por lo que no es posible. Vamos a eliminar el instituto antes insertado.

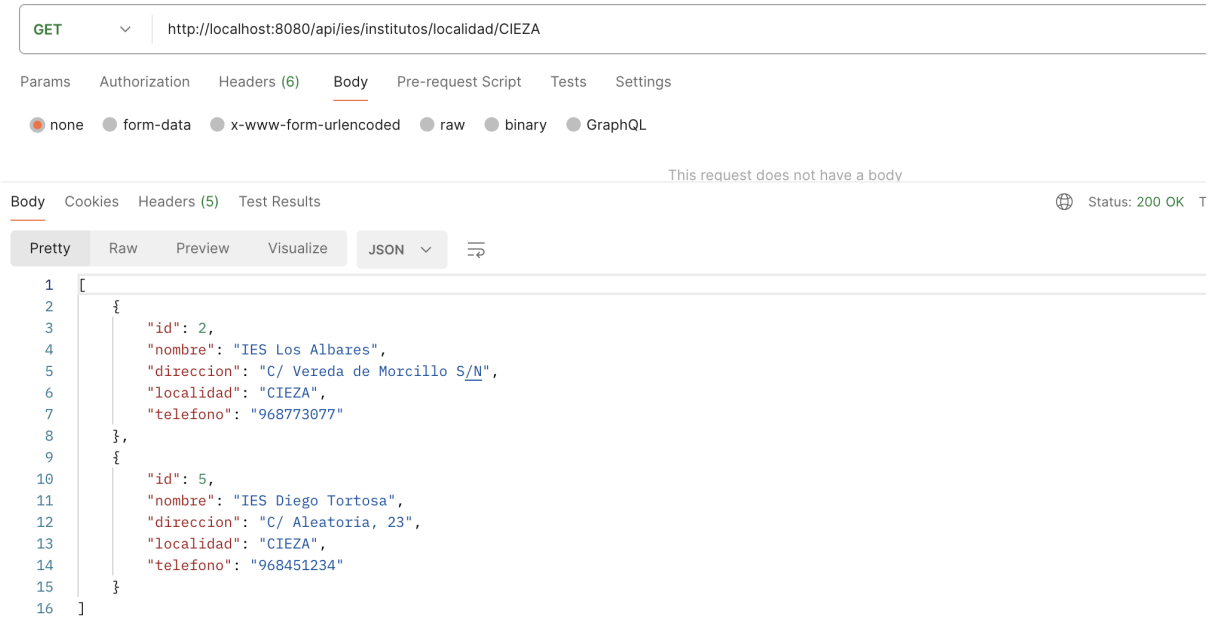


Finalmente vamos a probar estos dos métodos “extra” para probar estas dos bondades que incluye Spring Data Jpa (el uso de métodos automáticos si están nombrados de manera adecuada y JPQL).

Para que al probar ambos métodos se tenga más de un instituto de una localidad y más de un instituto con un prefijo inicial para probar añado un instituto de manera manual.



- Listar todos los institutos de un pueblo. Para ello se utilizará el endpoint http://localhost:8080/api/ies/institutos/localidad/{localidad}, en nuestro ejemplo utilizamos http://localhost:8080/api/ies/institutos/localidad/CIEZA



- Listar todos los institutos que comiencen por un prefijo. Para ello se utilizará el endpoint http://localhost:8080/api/ies/institutos/prefijo/{prefijo}, en nuestro ejemplo utilizamos http://localhost:8080/api/ies/institutos/prefijo/96845