

# MANUAL PROYECTO SPRING 2024

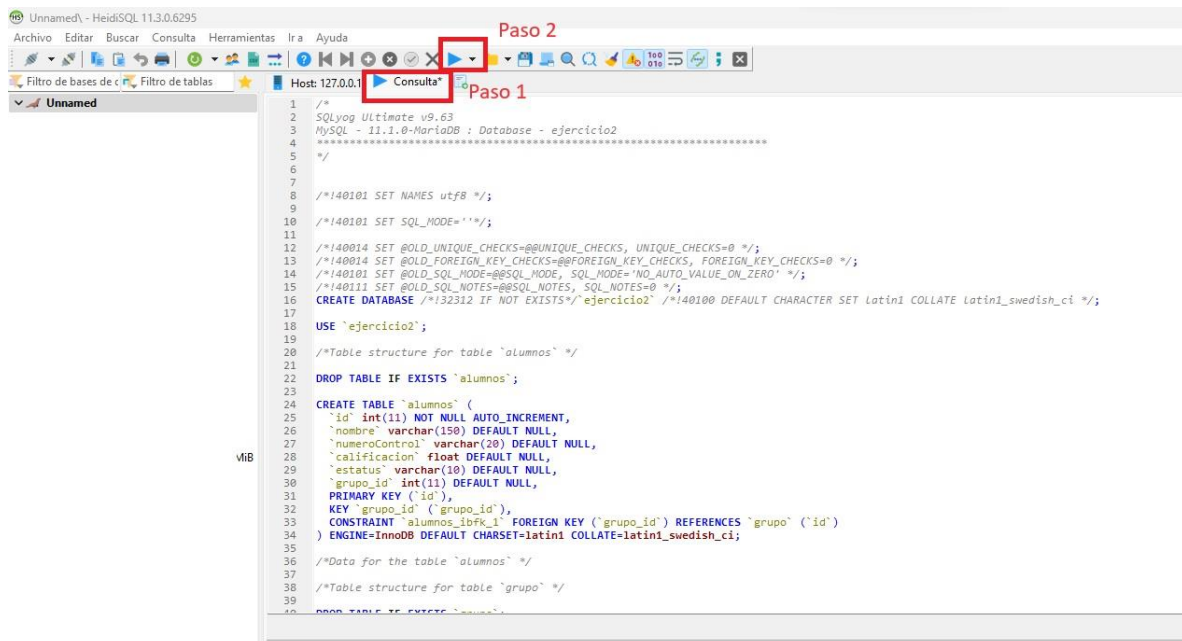
## INSTALACIÓN:

- Java JDK 17
- Spring Tools - <https://spring.io/tools>
- MariaDB - <https://mariadb.org/>
- Postman - <https://www.postman.com/>

## BASE DE DATOS:

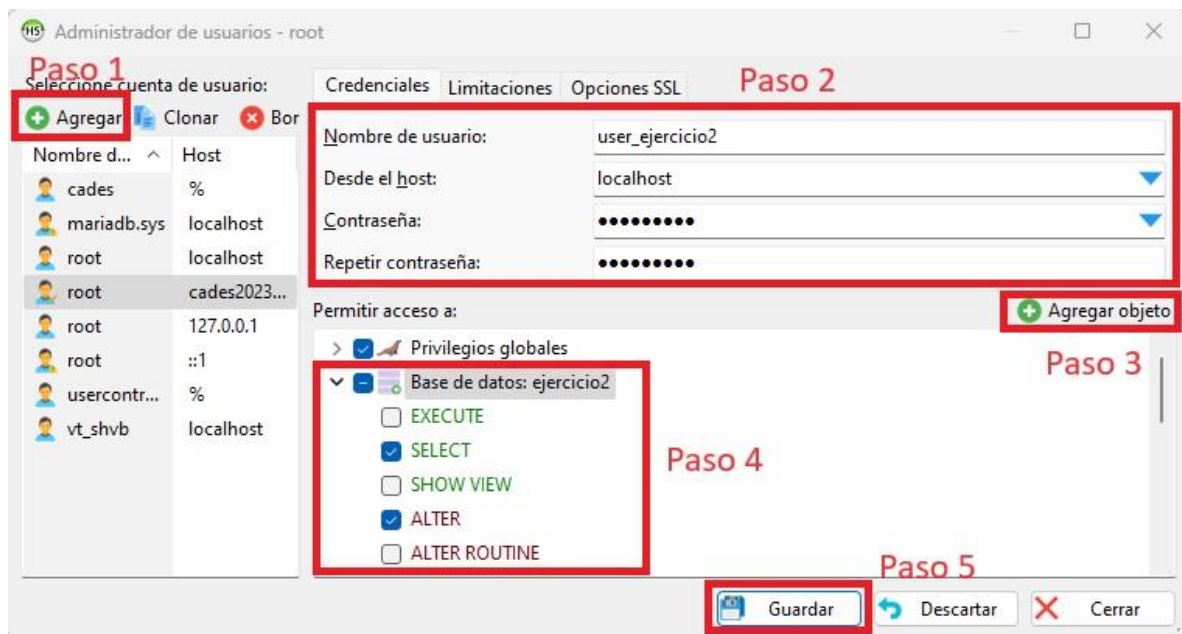
**Paso 1:** Descargar la base de datos, el script se encuentra en: <https://pastebin.com/raw/3YRMyrBB>

**Paso 2:** Abrir el programa HeidiSQL, vamos a la pestaña de Consulta (Paso 1) y luego damos clic para ejecutar la consulta (Paso 2) y con esto creamos nuestra base de datos



```
1  /*
2  SQLyog Ultimate v9.63
3  MySQL - 11.1.0-MariaDB : Database - ejercicio2
4  *****
5  */
6
7
8  /*!40101 SET NAMES utf8 */;
9
10 /*!40101 SET SQL_MODE='';*/
11
12 /*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
13 /*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
14 /*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
15 /*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;
16 CREATE DATABASE /*!32312 IF NOT EXISTS*/ `ejercicio2` /*!40100 DEFAULT CHARACTER SET latin1 COLLATE latin1_swedish_ci */;
17
18 USE `ejercicio2`;
19
20 /*Table structure for table `alumnos` */
21
22 DROP TABLE IF EXISTS `alumnos`;
23
24 CREATE TABLE `alumnos` (
25   `id` int(11) NOT NULL AUTO_INCREMENT,
26   `nombre` varchar(150) DEFAULT NULL,
27   `numeroControl` varchar(20) DEFAULT NULL,
28   `calificacion` float DEFAULT NULL,
29   `estatus` varchar(10) DEFAULT NULL,
30   `grupo_id` int(11) DEFAULT NULL,
31   PRIMARY KEY (`id`),
32   KEY `grupo_id` (`grupo_id`),
33   CONSTRAINT `alumnos_ibfk_1` FOREIGN KEY (`grupo_id`) REFERENCES `grupo` (`id`)
34 ) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_swedish_ci;
35
36 /*Data for the table `alumnos` */
37
38 /*Table structure for table `grupo` */
39
40 DROP TABLE IF EXISTS `grupo`;
```

**Paso 3:** Necesitamos generar un usuario para nuestra base de datos, para esto vamos al menú Herramientas -> Administrador de Usuarios nos desplegara una pantalla donde debemos dar clic en la opción de agregar (Paso 1), después escribimos el nombre del usuario, la contraseña y repetimos la contraseña (Paso 2), luego tenemos que dar clic en agregar Objeto(Paso 3), seleccionamos nuestra base de datos ejercicio2, y elegimos los permisos que vamos a necesitar en este caso que estamos en desarrollo seleccionamos: SELECT, ALTER, CREATE, DELETE, DROP, INSERT, UPDATE (Paso 4) y guardamos (Paso 5) y damos clic en el botón de cerrar



## APLICACIÓN:

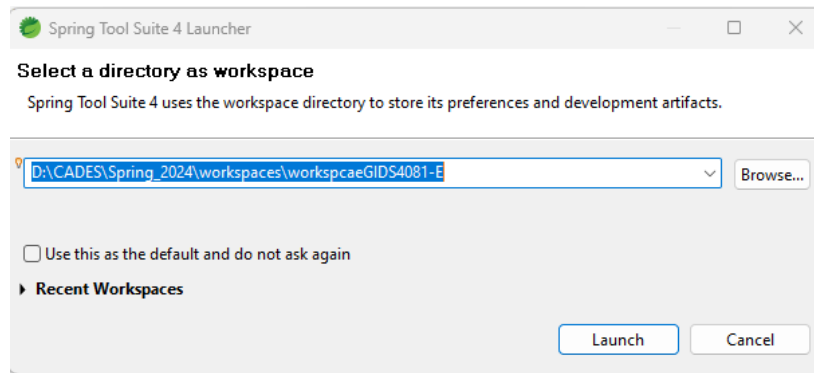
**Paso 1:** Nos posicionamos en la carpeta donde tenemos el archivo .jar que descargamos de la página de Spring y abrimos una terminal y ejecutamos el comando: `java -jar <<nombre del archivo .jar>>` y esperamos a que se descomprima el archivo

```
Windows PowerShell
PS C:\Users\sergi\Downloads> java -jar .\spring-tool-suite-4-21.1.RELEASE-e4.30.0-win32.win32.x86_64.self-extracting.jar
```

**Paso 2:** Vamos a la carpeta que se generó y ejecutamos el archivo SpringToolSuite4.exe

configuration	06/02/2024 03:30 p. m.	Carpeta de archivos	
dropins	19/01/2024 10:52 a. m.	Carpeta de archivos	
features	19/01/2024 10:53 a. m.	Carpeta de archivos	
p2	19/01/2024 10:53 a. m.	Carpeta de archivos	
plugins	19/01/2024 10:54 a. m.	Carpeta de archivos	
readme	19/01/2024 10:54 a. m.	Carpeta de archivos	
.eclipseproduct	19/01/2024 10:53 a. m.	Archivo ECLIPSEP...	1 KB
artifacts	19/01/2024 10:53 a. m.	Archivo XML	151 KB
license	19/01/2024 10:53 a. m.	Archivo TXT	12 KB
open-source-licenses	19/01/2024 10:53 a. m.	Archivo TXT	612 KB
SpringToolSuite4	19/01/2024 10:53 a. m.	Aplicación	521 KB
SpringToolSuite4	19/01/2024 10:52 a. m.	Opciones de confi...	1 KB
SpringToolSuite4c	19/01/2024 10:53 a. m.	Aplicación	233 KB

**Paso 3:** Una vez que se ejecute, nos pedirá una carpeta para crear el espacio de trabajo y damos clic en el botón Launch



**Paso 4:** Una vez que se ejecute el Spring Tools, crearemos un nuevo proyecto seleccionando la opción de Create new Spring Starter Project esto nos abrirá una nueva pestaña donde modificaremos los siguientes datos:

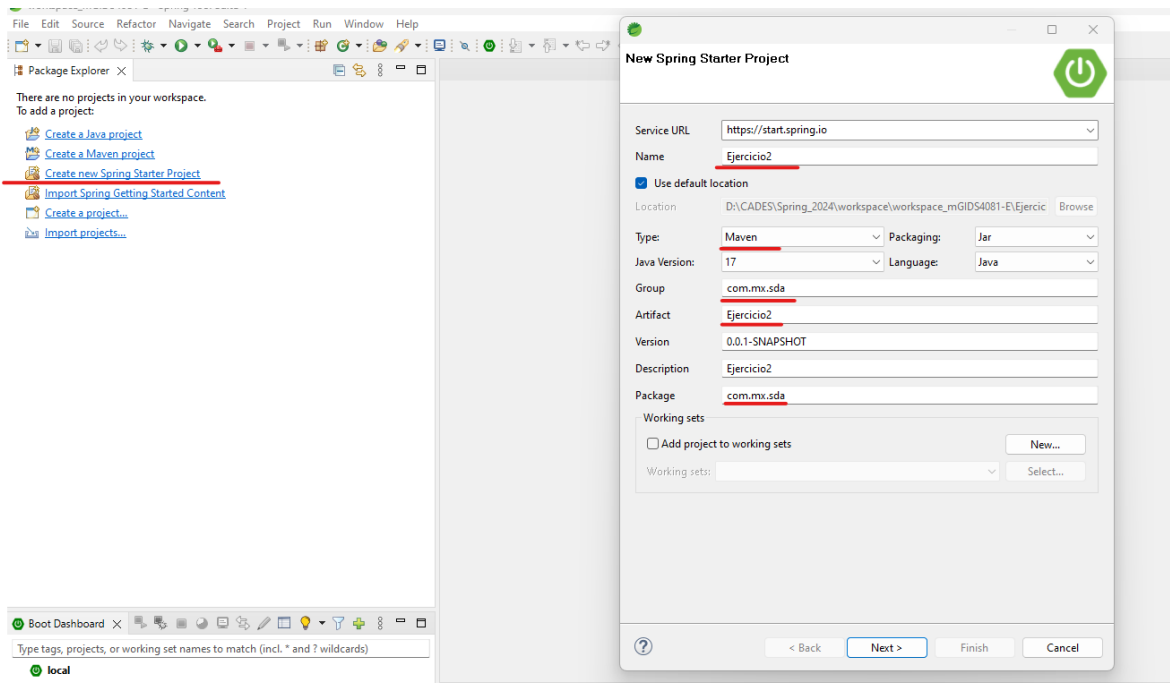
Name: Ejercicio2

Type: Maven

Group: com.mx.sda

Artifac: Ejercicio2\_

Package: com.mx.sda



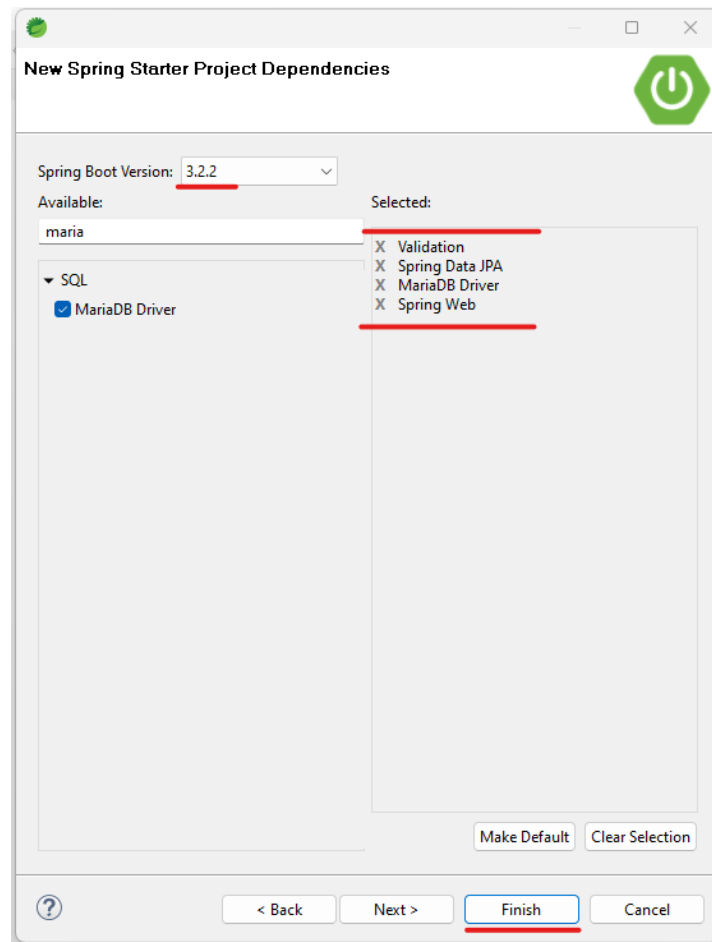
Los demás datos los podemos dejar como están y damos clic en **Next>** , seleccionamos las siguientes dependencias:

Validation

Spring Data Jpa

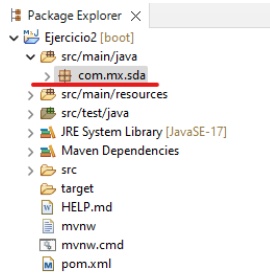
MariaDB Driver o MySQL Driver

Spring Web



Y damos clic en **Finish** y esperamos a que se genere nuestro proyecto.

**Paso 5:** Una vez que se genera el proyecto, vamos al paquete principal y crearemos los paquetes que vamos a necesitar:



Damos clic derecho sobre el paquete y damos clic en New -> Package, los paquetes que vamos a crear son:

com.mx.sda.controllers

com.mx.sda.dto

com.mx.sda.exception

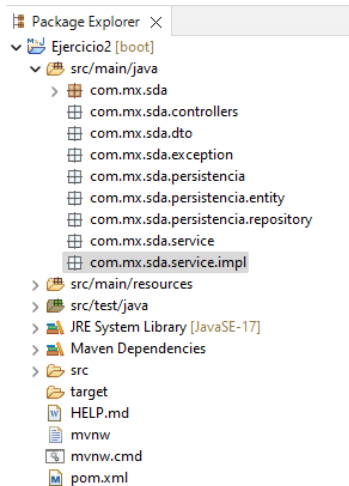
com.mx.sda.persistencia

com.mx.sda.persistencia.entity

com.mx.sda.persistencia.repository

com.mx.sda.service

com.mx.sda.service.impl



**Paso 6:** Vamos a crear las clases Entity que vamos a necesitar, nos vamos al paquete com.mx.sda.persistencia.entity damos clic derecho New -> Class -> Alumnos

```

package com.mx.sda.persistencia.entity;

import jakarta.persistence.Entity;

@Entity
public class Alumnos {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String nombre;
    private String numeroControl;
    private Float calificacion;

    @Enumerated(EnumType.STRING)
    private AlumnoEstatus estatus;

    @ManyToOne
    @JoinColumn(name="grupo_id")
    private Grupo grupo;

    public static enum AlumnoEstatus{
        REGULAR, BAJA_TEMPORAL
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getNumeroControl() {
        return numeroControl;
    }

    public void setNumeroControl(String numeroControl) {
        this.numeroControl = numeroControl;
    }

    public Float getCalificacion() {
        return calificacion;
    }
}

```

Hacemos el mismo proceso y ahora creamos la clase Grupo

```

package com.mx.sda.persistencia.entity;

import jakarta.persistence.Entity;
import jakarta.persistence.EnumType;
import jakarta.persistence.Enumerated;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
public class Grupo {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    private String nombre;

    @Enumerated(EnumType.STRING)
    private GrupoEstatus estatus;

    public static enum GrupoEstatus {
        HABILITADO, DESHABILITADO
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public GrupoEstatus getEstatus() {
        return estatus;
    }

    public void setEstatus(GrupoEstatus estatus) {
        this.estatus = estatus;
    }

}

```

**Paso 7:** Vamos a crear las clases Dto que vamos a necesitar, nos vamos al paquete com.mx.sda.dto damos clic derecho New -> Class -> AlumnosDto

```

package com.mx.sda.dto;

import jakarta.validation.constraints.DecimalMin;

public class AlumnosDto {
    @NotBlank
    private String nombre;

    @NotBlank
    private String numeroControl;

    @DecimalMin(value = "0.01")
    private Float calificacion;

    @Min(value = 1)
    private Integer grupo_id;

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getNumeroControl() {
        return numeroControl;
    }

    public void setNumeroControl(String numeroControl) {
        this.numeroControl = numeroControl;
    }

    public Float getCalificacion() {
        return calificacion;
    }

    public void setCalificacion(Float calificacion) {
        this.calificacion = calificacion;
    }

    public Integer getGrupo_id() {
        return grupo_id;
    }

    public void setGrupo_id(Integer grupo_id) {
        this.grupo_id = grupo_id;
    }
}

```

Hacemos el mismo proceso y ahora creamos la clase GrupoDto



```

package com.mx.sda.dto;

import jakarta.validation.constraints.NotBlank;

public class GrupoDto {
    @NotBlank
    private String nombre;

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}

```

**Paso 8:** Vamos a crear las clases Service que vamos a necesitar, nos vamos al paquete com.mx.sda.service damos clic derecho New -> Interface -> AlumnosService

```

package com.mx.sda.service;

import java.util.Optional;

public interface AlumnosService {

    Page<Alumnos> findAll(Pageable pageable);

    Optional<Alumnos> findOneById(Integer alumnoId);

    Alumnos createOne(AlumnosDto alumnosDto);

    Alumnos updateOneById(Integer alumnoId, AlumnosDto alumnosDto);

    Alumnos disableOneById(Integer alumnoId);
}

```

Hacemos el mismo proceso y ahora creamos la interface GrupoService

```

package com.mx.sda.service;

import java.util.Optional;

public interface GrupoService {

    Page<Grupo> findAll(Pageable pageable);

    Optional<Grupo> findOneById(Integer grupoId);

    Grupo createOne(GrupoDto grupoDto);

    Grupo updateOneById(Integer grupoId, GrupoDto grupoDto);

    Grupo disableOneById(Integer grupoId);
}

```

**Paso 9:** Vamos a crear las clases controller que vamos a necesitar, nos vamos al paquete `com.mx.sda.controllers` damos clic derecho New -> Class -> `AlumnosController`

```
import com.mx.sda.dto.AlumnosDto;
import com.mx.sda.persistencia.entity.Alumnos;
import com.mx.sda.service.AlumnosService;

import jakarta.validation.Valid;

@RestController
@RequestMapping("/alumnos")
public class AlumnosController {
    @Autowired
    private AlumnosService alumnosService;

    @GetMapping
    public ResponseEntity<Page<Alumnos>> findAll(Pageable pageable) {
        Page<Alumnos> alumnosPage = alumnosService.findAll(pageable);
        if (alumnosPage.hasContent()) {
            return ResponseEntity.ok(alumnosPage);
        }
        return ResponseEntity.notFound().build();
    }

    @GetMapping("/{alumnoId}")
    public ResponseEntity<Alumnos> findById(@PathVariable Integer alumnoId) {
        Optional<Alumnos> alumnos = alumnosService.findById(alumnoId);
        if (alumnos.isPresent()) {
            return ResponseEntity.ok(alumnos.get());
        }
        return ResponseEntity.notFound().build();
    }

    @PostMapping
    public ResponseEntity<?> createOne(@RequestBody @Valid AlumnosDto alumnosDto) {
        Alumnos alumnos = alumnosService.createOne(alumnosDto);
        return ResponseEntity.status(HttpStatus.CREATED).body(alumnos);
    }

    @PutMapping("/{alumnosId}")
    public ResponseEntity<?> updateOneById(@PathVariable Integer alumnosId,
        @RequestBody @Valid AlumnosDto saveAlumnos) {
        Alumnos alumnos = alumnosService.updateOneById(alumnosId, saveAlumnos);
        return ResponseEntity.ok(alumnos);
    }

    @PutMapping("/{alumnosId}/disabled")
    public ResponseEntity<Alumnos> disableOneById(@PathVariable Integer alumnosId) {
        Alumnos alumnos = alumnosService.disableOneById(alumnosId);
        return ResponseEntity.ok(alumnos);
    }
}
```

Hacemos el mismo proceso y ahora creamos la clase `GrupoController`

```

package com.mx.sda.controllers;

import java.util.Optional;

@RestController
@RequestMapping("/grupos")
public class GrupoController {

    @Autowired
    private GrupoService grupoService;

    @GetMapping
    public ResponseEntity<Page<Grupo>> findAll(Pageable pageable) {
        Page<Grupo> gruposPage = grupoService.findAll(pageable);
        if (gruposPage.hasContent()) {
            return ResponseEntity.ok(gruposPage);
        }
        return ResponseEntity.notFound().build();
    }

    @GetMapping("/{grupoId}")
    public ResponseEntity<Grupo> findOneById(@PathVariable Integer grupoId) {
        Optional<Grupo> grupos = grupoService.findOneById(grupoId);
        if (grupos.isPresent()) {
            return ResponseEntity.ok(grupos.get());
        }
        return ResponseEntity.notFound().build();
    }

    @PostMapping
    public ResponseEntity<Grupo> createOne(@RequestBody @Valid GrupoDto grupoDto) {
        Grupo grupo = grupoService.createOne(grupoDto);
        return ResponseEntity.status(HttpStatus.CREATED).body(grupo);
    }

    @PutMapping("/{grupoId}")
    public ResponseEntity<Grupo> updateOneById(@PathVariable Integer grupoId, @RequestBody @Valid GrupoDto grupoDto) {
        Grupo grupo = grupoService.updateOneById(grupoId, grupoDto);
        return ResponseEntity.ok(grupo);
    }

    @PutMapping("/{grupoId}/disabled")
    public ResponseEntity<Grupo> disableOneById(@PathVariable Integer grupoId) {
        Grupo grupo = grupoService.disableOneById(grupoId);
        return ResponseEntity.ok(grupo);
    }
}

```

**Paso 10:** Vamos a crear la clase exception, nos vamos al paquete com.mx.sda.exception damos clic derecho New -> Class -> ObjectNotFoundException

```

package com.mx.sda.exception;

public class ObjectNotFoundException extends RuntimeException {

    public ObjectNotFoundException() {
    }

    public ObjectNotFoundException(String message) {
        super(message);
    }

    public ObjectNotFoundException(String message, Throwable cause) {
        super(message, cause);
    }
}

```

**Paso 11:** Vamos a crear las clases repository que vamos a necesitar, nos vamos al paquete com.mx.sda.persistencia.repository damos clic derecho New -> Interface -> AlumnosRepository

```
package com.mx.sda.persistencia.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.mx.sda.persistencia.entity.Alumnos;

public interface AlumnosRepository extends JpaRepository<Alumnos, Integer> {

}
```

Hacemos el mismo proceso y ahora creamos la interface GrupoRepository

```
package com.mx.sda.persistencia.repository;

import org.springframework.data.jpa.repository.JpaRepository;

public interface GrupoRepository extends JpaRepository<Grupo, Integer> {

}
```

**Paso 12:** Vamos a crear las clases para implementar las interfaces Service, nos vamos al paquete com.mx.sda.service.impl damos clic derecho New -> Class -> AlumnosserviceImpl

```
package com.mx.sda.service.impl;

import java.util.Optional;

@Service
public class AlumnosServiceImpl implements AlumnosService {

    @Autowired
    private AlumnosRepository alumnosRepository;

    @Override
    public Page<Alumnos> findAll(Pageable pageable) {
        return alumnosRepository.findAll(pageable);
    }

    @Override
    public Optional<Alumnos> findOneById(Integer alumnoId) {
        return alumnosRepository.findById(alumnoId);
    }

    @Override
    public Alumnos createOne(AlumnosDto alumnosDto) {
        Alumnos alumnos = new Alumnos();
        alumnos.setNombre(alumnosDto.getNombre());
        alumnos.setNumeroControl(alumnosDto.getNumeroControl());
        alumnos.setCalificacion(alumnosDto.getCalificacion());
        alumnos.setEstatus(AlumnoEstatus.REGULAR);

        Grupo grupo = new Grupo();
        grupo.setId(alumnosDto.getGrupo_id());

        alumnos.setGrupo(grupo);

        return alumnosRepository.save(alumnos);
    }
}
```

```

    @Override
    public Alumnos updateOneById(Integer alumnoId, AlumnosDto alumnosDto) {
        Alumnos alumnoDB = alumnosRepository.findById(alumnoId)
            .orElseThrow(() -> new ObjectNotFoundException("No se encuentra el alumno con el id: " + alumnoId));

        alumnoDB.setNombre(alumnosDto.getNombre());
        alumnoDB.setNumeroControl(alumnosDto.getNumeroControl());
        alumnoDB.setCalificacion(alumnosDto.getCalificacion());

        Grupo grupo = new Grupo();
        grupo.setId(alumnosDto.getGrupo_id());

        alumnoDB.setGrupo(grupo);

        return alumnosRepository.save(alumnoDB);
    }

    @Override
    public Alumnos disableOneById(Integer alumnoId) {
        Alumnos alumnoDB = alumnosRepository.findById(alumnoId)
            .orElseThrow(() -> new ObjectNotFoundException("No se encuentra el alumno con el id: " + alumnoId));
        alumnoDB.setEstatus(AlumnoEstatus.BAJA_TEMPORAL);

        return alumnosRepository.save(alumnoDB);
    }
}

```

Hacemos el mismo proceso y ahora creamos la clase GrupoServiceImpl

```

package com.mx.sda.service.impl;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.stereotype.Service;

import com.mx.sda.dto.GrupoDto;
import com.mx.sda.exception.ObjectNotFoundException;
import com.mx.sda.persistencia.entity.Grupo;
import com.mx.sda.persistencia.entity.Grupo.GrupoEstatus;
import com.mx.sda.persistencia.repository.GrupoRepository;
import com.mx.sda.service.GrupoService;

@Service
public class GrupoServiceImpl implements GrupoService {

    @Autowired
    private GrupoRepository grupoRepository;

    @Override
    public Page<Grupo> findAll(Pageable pageable) {
        return grupoRepository.findAll(pageable);
    }

    @Override
    public Optional<Grupo> findOneById(Integer groupId) {
        return grupoRepository.findById(groupId);
    }

    @Override
    public Grupo createOne(GrupoDto grupoDto) {
        Grupo grupo = new Grupo();
        grupo.setNombre(grupoDto.getNombre());
        grupo.setEstatus(GrupoEstatus.HABILITADO);

        return grupoRepository.save(grupo);
    }
}

```

```

@Override
public Grupo updateOneById(Integer groupId, GrupoDto grupoDto) {
    Grupo alumnoDB = grupoRepository.findById(groupId)
        .orElseThrow(() -> new ObjectNotFoundException("No se encuentra el grupo con el id: " + groupId));

    alumnoDB.setNombre(grupoDto.getNombre());

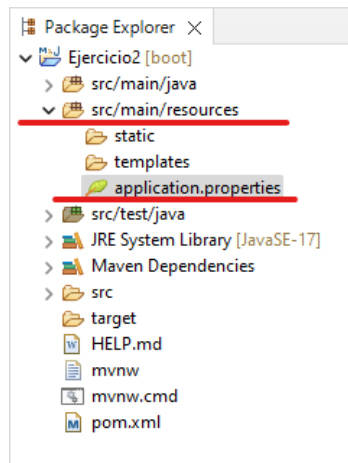
    return grupoRepository.save(alumnoDB);
}

@Override
public Grupo disableOneById(Integer groupId) {
    Grupo alumnoDB = grupoRepository.findById(groupId)
        .orElseThrow(() -> new ObjectNotFoundException("No se encuentra el grupo con el id: " + groupId));
    alumnoDB.setEstatus(GrupoEstatus.DESHABILITADO);

    return grupoRepository.save(alumnoDB);
}
}

```

**Paso 13:** Ahora vamos a configurar la conexión con la base de datos, para esto nos a src/main/resources al archivo application.properties



Si tienen instalado Mysql la conexión seria de la siguiente manera:

```

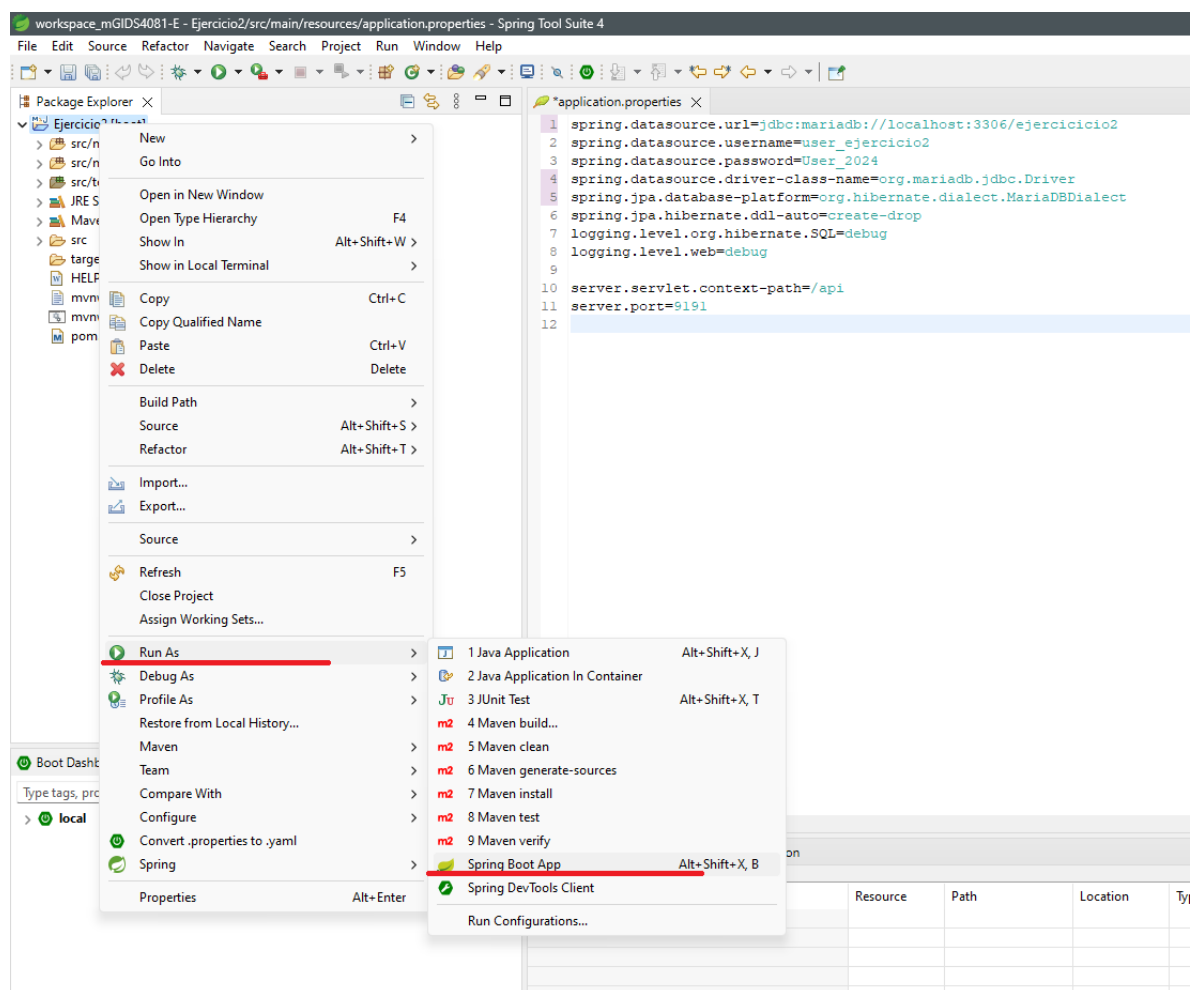
application.properties
1 spring.datasource.url=jdbc:mysql://localhost:3306/ejercicicio2
2 spring.datasource.username=user_ejercicio2
3 spring.datasource.password=User_2024
4 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
5 spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
6 spring.jpa.hibernate.ddl-auto=create-drop
7 logging.level.org.hibernate.SQL=debug
8 logging.level.web=debug
9
10 server.servlet.context-path=/api
11 server.port=9191

```

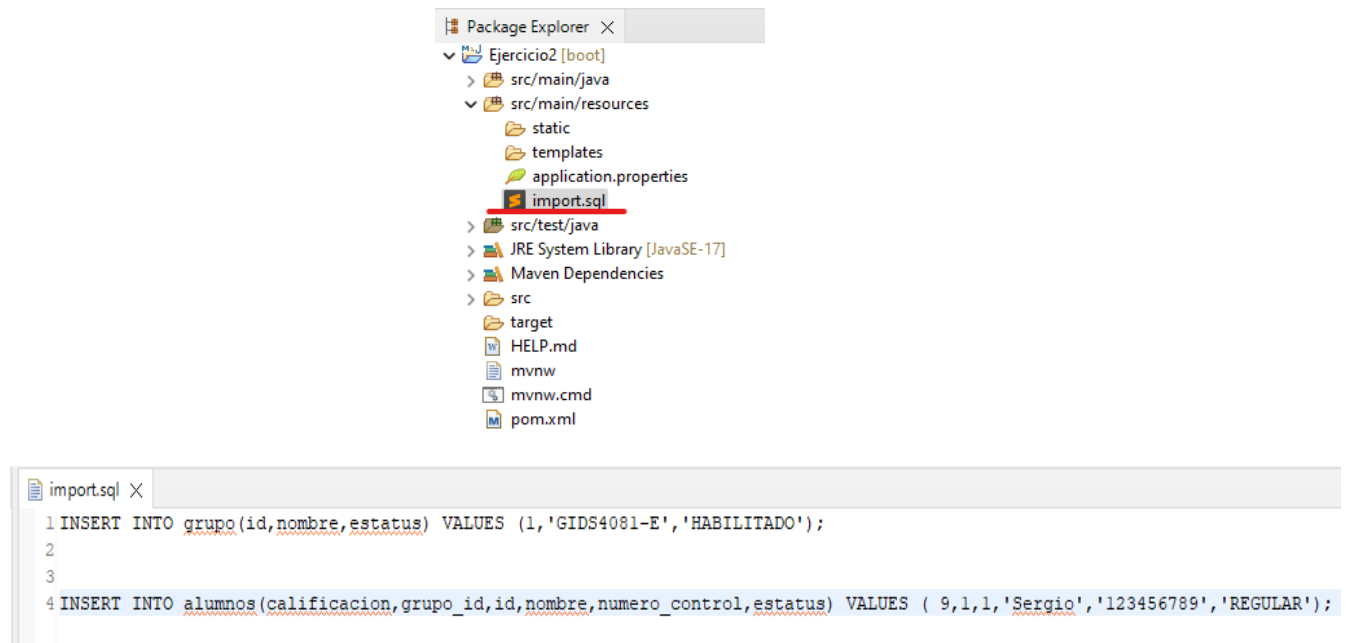
Si tienen instalado MariaDB la conexión sería de la siguiente manera:

```
*application.properties X
1 spring.datasource.url=jdbc:mariadb://localhost:3306/ejercicicio2
2 spring.datasource.username=user_ejercicio2
3 spring.datasource.password=User_2024
4 spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
5 spring.jpa.database-platform=org.hibernate.dialect.MariaDBDialect
6 spring.jpa.hibernate.ddl-auto=create-drop
7 logging.level.org.hibernate.SQL=debug
8 logging.level.web=debug
9
10 server.servlet.context-path=/api
11 server.port=9191
```

**Paso 14:** Para ejecutar nuestra aplicación, damos clic derecho sobre nuestro proyecto -> Run As -> Spring Boot App



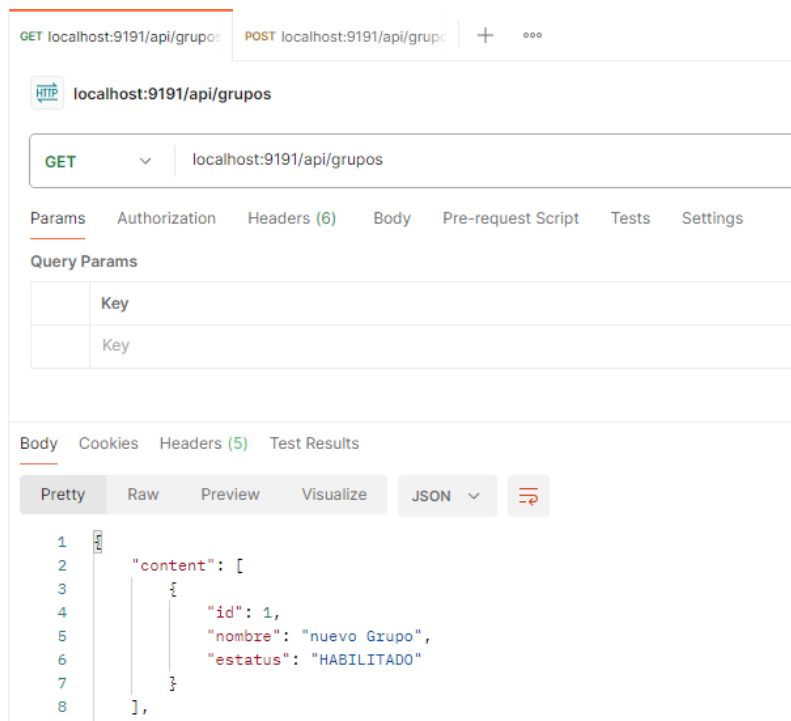
**Paso 15:** Ahora cada vez que se inicie nuestro proyecto nos borrara la información que se encuentren en la base de datos, para tener datos al iniciar nuestro proyecto, agregamos un archivo en src/main/resources llamado import.sql



**Paso 16:** Para probar los métodos abrimos el postman:

Método: GET

URL: localhost:9191/api/grupos

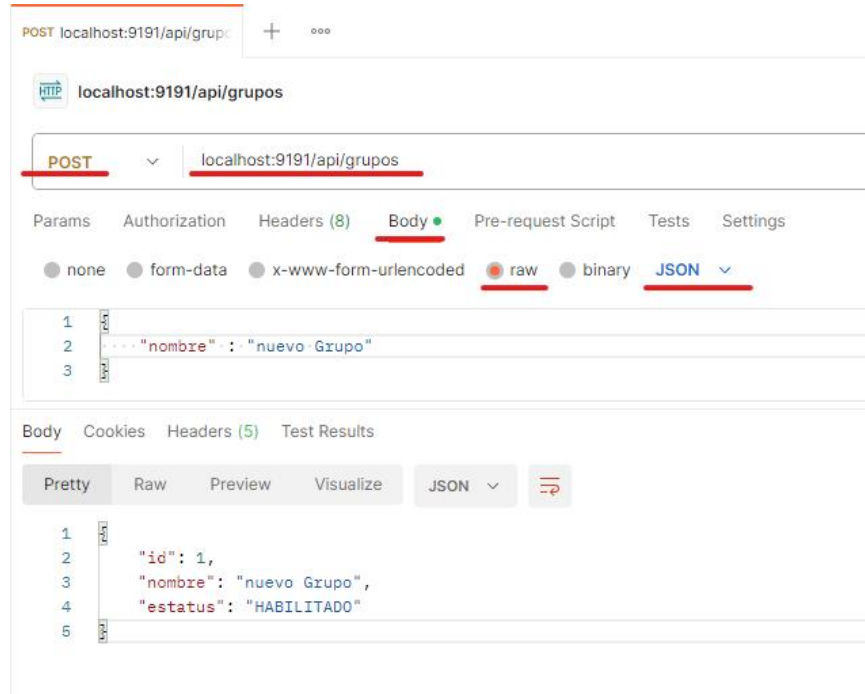




Método: POST

URL: localhost:9191/api/grupos

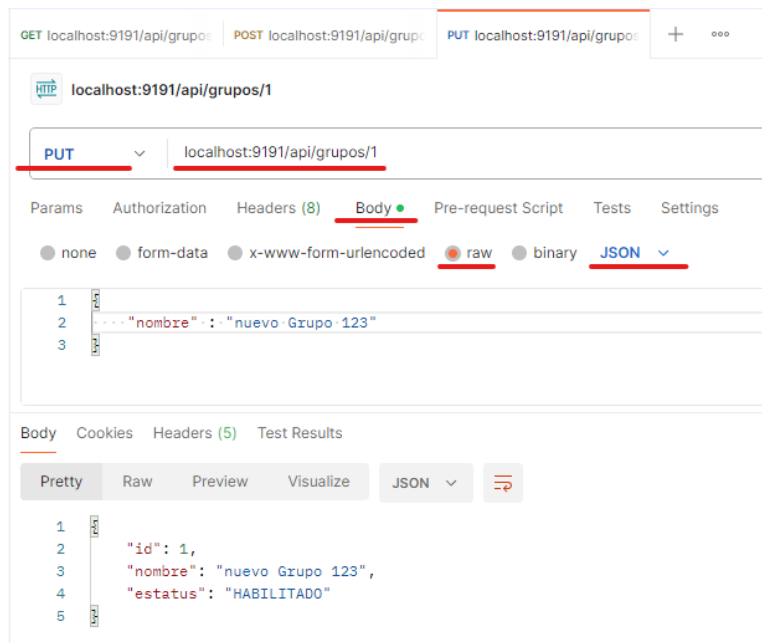
Body -> raw -> JSON



Método: PUT

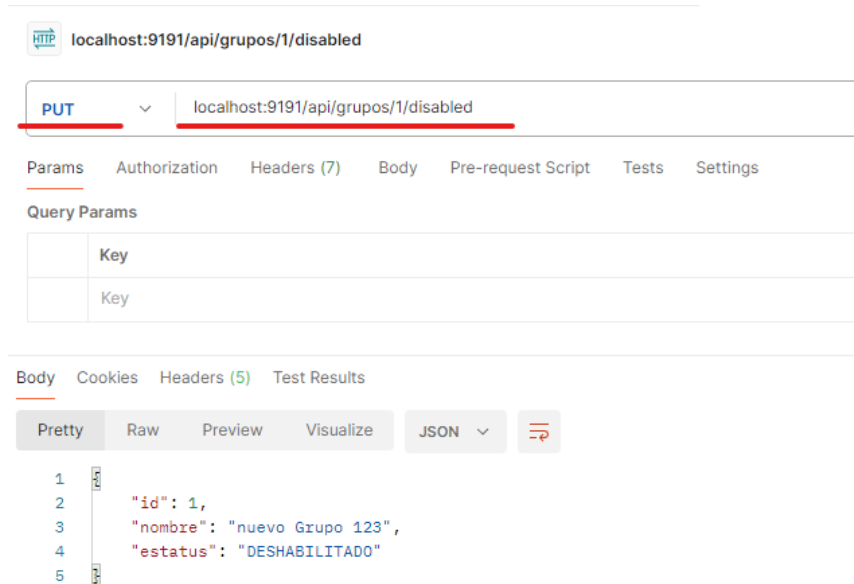
URL: localhost:9191/api/grupos /1

Body -> raw -> JSON



Método: PUT

URL: localhost:9191/api/grupos /1/disabled



localhost:9191/api/grupos/1/disabled

PUT localhost:9191/api/grupos/1/disabled

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

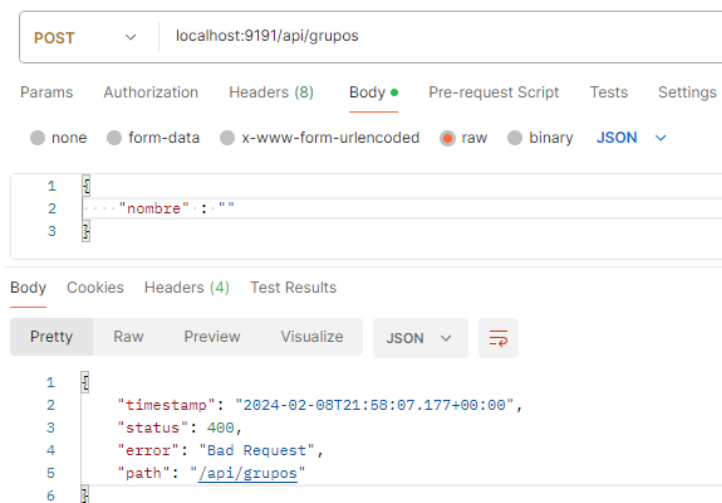
Key
Key

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "nombre": "nuevo Grupo 123",
4   "estatus": "DESHABILITADO"
5 }
```

**Paso 17:** Ahora vamos a agregar las validaciones, porque si hacemos un POST y dejamos vacío el campo de nombre, nos manda el error, pero no especifica que sucedió



POST localhost:9191/api/grupos

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

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

```
1 {
2   "nombre": ""
3 }
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "timestamp": "2024-02-08T21:58:07.177+00:00",
3   "status": 400,
4   "error": "Bad Request",
5   "path": "/api/grupos"
6 }
```

**Paso 18:** Para agregar la validación nos vamos a las clases del paquete controller abrimos la clase GrupoController y vamos al método createOne y al método updateOneById para agregar la validación

```

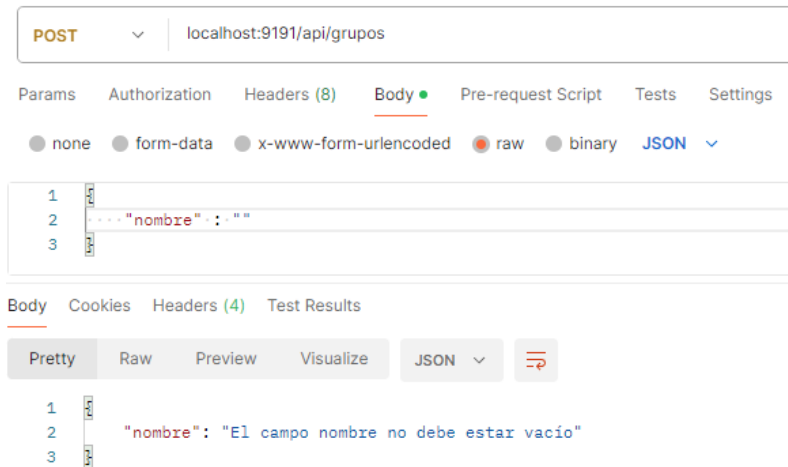
@PostMapping
public ResponseEntity<?> createOne(@RequestBody @Valid GrupoDto grupoDto, BindingResult result) {
    if (result.hasFieldErrors()) {
        return validation(result);
    }
    Grupo grupo = grupoService.createOne(grupoDto);
    return ResponseEntity.status(HttpStatus.CREATED).body(grupo);
}

@PutMapping("/{grupoId}")
public ResponseEntity<?> updateOneById(@PathVariable Integer grupoId, @RequestBody @Valid GrupoDto grupoDto,
    BindingResult result) {
    if (result.hasFieldErrors()) {
        return validation(result);
    }
    Grupo grupo = grupoService.updateOneById(grupoId, grupoDto);
    return ResponseEntity.ok(grupo);
}

private ResponseEntity<?> validation(BindingResult result) {
    Map<String, String> errores = new HashMap<String, String>();
    result.getFieldErrors().forEach(err -> {
        errores.put(err.getField(), "El campo " + err.getField() + " " + err.getDefaultMessage());
    });
    return ResponseEntity.badRequest().body(errores);
}

```

**Paso 19:** Ahora si nuevamente hacemos un POST y dejamos vacío el campo de nombre, nos manda el error específico de lo que sucedió



**Paso 20:** Si requerimos modificar las validaciones se harían desde la clase para este caso GrupoDto, por ejemplo, que el nombre grupo debe tener entre 3 y 10 caracteres.

```

package com.mx.sda.dto;

import jakarta.validation.constraints.Size;

public class GrupoDto {
    @Size(min = 3, max = 10)
    private String nombre;

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}

```

Si lo vemos lo revisamos en el postman nos debería regresar lo siguiente:

The screenshot shows the Postman interface for a POST request to `localhost:9191/api/grupos`. The request body is a JSON object: `{ "nombre": "" }`. The response body is a JSON object: `{ "nombre": "El campo nombre el tamaño debe estar entre 3 y 10" }`.

```

POST localhost:9191/api/grupos

{
  "nombre": ""
}

{
  "nombre": "El campo nombre el tamaño debe estar entre 3 y 10"
}

```