# Desarrollo rápido de aplicaciones.

## Actividad Spring Boot.

Jesús Fornieles Muñoz

# Índice:

# 3. Actividades

## 3.1. Crear servicios REST para la entidad Hero

Se ha seguido el tutorial de Angular "Tour of Heroes" adaptando la parte de backend con Spring Boot para crear un servicio REST que suministre la información de los héroes desde una base de datos PostgreSQL.

#### Entidad `Hero`

Se creó la entidad `Hero` que incluye:

- `id` (Long)
- `name` (String)

#### Añadir entidad `Power`

Se ha creado la entidad `Power` (Superpoder), relacionada con `Hero` mediante la anotación `@OneToMany`.

Relación utilizada:

```java
@OneToMany(cascade = CascadeType.ALL, orphanRemoval = true)
@JoinColumn(name = "hero_id")
private List<Power> powers;
```

Entidad heroe:

```java
package com.example.demo.entity;

import jakarta.persistence.*;
import java.util.List;

@Entity
public class Hero {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private String name;

    @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true)
    @JoinColumn(name = "hero_id")
    private List<Power> powers;

    public Hero() {}

    public Hero(String name, List<Power> powers) {
        this.name = name;
        this.powers = powers;
    }

    public Long getId() {
        return id;
    }

    public String getName() {
        return name;
    }
    public List<Power> getPowers() {
        return powers;
    }

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

    public void setName(String name) {
        this.name = name;
    }

    public void setPowers(List<Power> powers) {
        this.powers = powers;
    }
}
```

Entidad Power:

```java
package com.example.demo.entity;

import jakarta.persistence.*;

@Entity
public class Power {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private String name;

    public Power() {}

    public Power(String name) {
        this.name = name;
    }

    public Long getId() {
        return id;
    }

    public String getName() {
        return name;
    }

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

    public void setName(String name) {
        this.name = name;
    }
}
```

### Controlador `HeroController`

Se expusieron los siguientes endpoints:

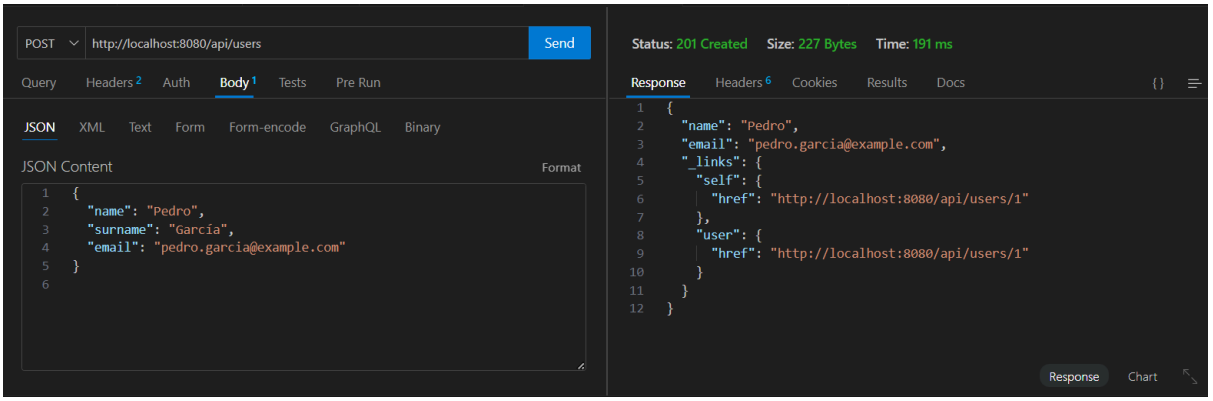
- `GET /api/heroes` — Obtener todos los héroes.
- `POST /api/heroes` — Crear un nuevo héroe con lista de superpoderes.

```java
package com.example.demo.controller;

import com.example.demo.dto.HeroDTO;
import com.example.demo.entity.Hero;
import com.example.demo.mapper.HeroMapper;
import com.example.demo.repository.HeroRepository;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.stream.Collectors;

@RestController
@RequestMapping("/heroes")
public class HeroController {

    private final HeroRepository heroRepository;
    private final HeroMapper heroMapper;

    public HeroController(HeroRepository heroRepository, HeroMapper heroMapper) {
        this.heroRepository = heroRepository;
        this.heroMapper = heroMapper;
    }

    @GetMapping
    public List<HeroDTO> getAllHeroes() {
        return heroRepository.findAll().stream()
                .map(heroMapper::toDto)
                .collect(Collectors.toList());
    }

    @PostMapping
    public HeroDTO createHero(@RequestBody HeroDTO heroDTO) {
        Hero hero = heroMapper.toEntity(heroDTO);
        Hero savedHero = heroRepository.save(hero);
        return heroMapper.toDto(savedHero);
    }
}
```

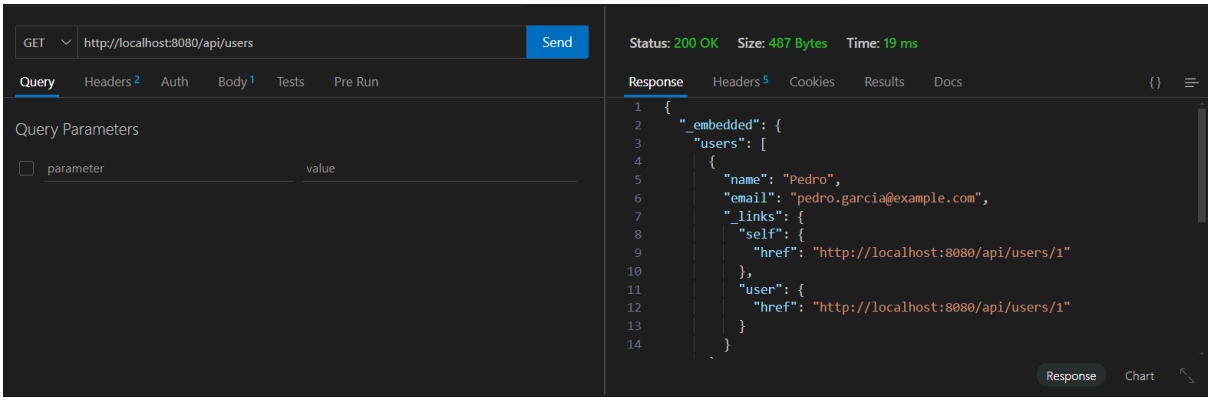
**Pruebas de funcionamiento:**

Vamos a utilizar thunder Client, que es una extensión de vscode que nos permite probar apis de forma sencilla:

## Post:




## Get:



# 3.2. Crear un endpoint personalizado

Se ha creado un endpoint personalizado para buscar usuarios mediante su apellido.

**Repositorio `UserRepository`**

Se añadió el método personalizado en la interfaz del repositorio:

```java
package com.example.demo.repository;

import com.example.demo.entity.User;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface UserRepository extends CrudRepository<User, Long> {
    List<User> findBySurname(String surname);
}
```

**Controlador `UserController`**

Se creó el controlador REST que expone:

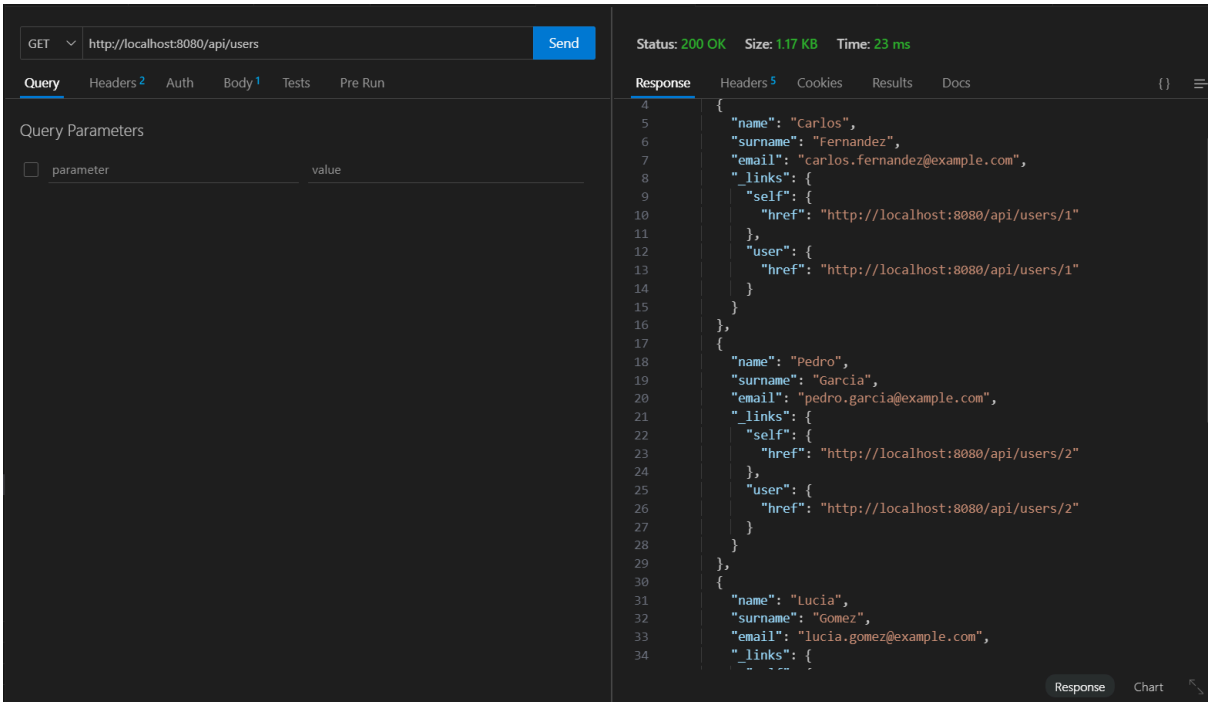
- `GET /api/users` — Obtener todos los usuarios.
- `POST /api/users` — Crear un nuevo usuario.
- `GET /api/users/search?surname={surname}` — Buscar usuario por apellido.

```java
package com.example.demo.controller;

import com.example.demo.entity.User;
import com.example.demo.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/users")
public class UserController {

    @Autowired
    private UserRepository userRepository;
    @GetMapping
    public List<User> getAllUsers() {
        return (List<User>) userRepository.findAll();
    }

    @PostMapping
    public User createUser(@RequestBody User user) {
        return userRepository.save(user);
    }

    @GetMapping("/search")
    public List<User> searchUsersBySurname(@RequestParam String surname) {
        return userRepository.findBySurname(surname);
    }
}
```

**Pruebas de funcionamiento**

Vamos a añadir un par de ejemplos de usuarios más al endpoint de users para hacer una búsqueda por apellido:

Estado inicial:




Endpoint personalizado de busqueda por apellido: