ucm

Programación de Aplicaciones Web

Electivo III

Román Gajardo

https://github.com/romanncodes

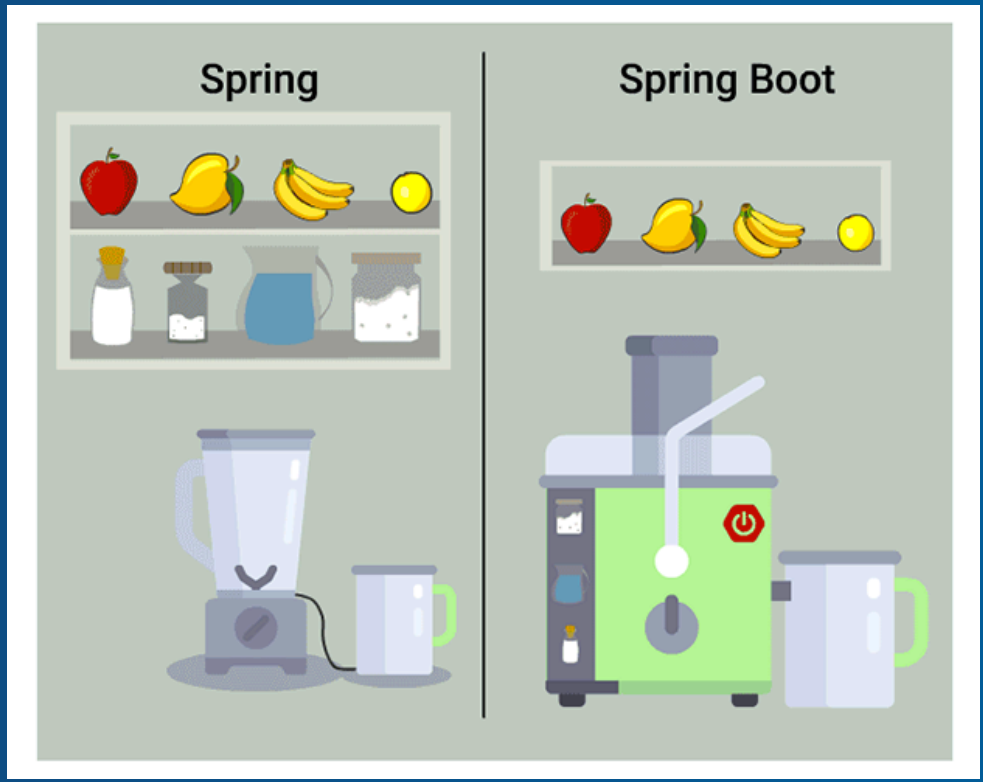
roman.gajardo@gmail.com

Contenidos

- Java
- Spring Boot

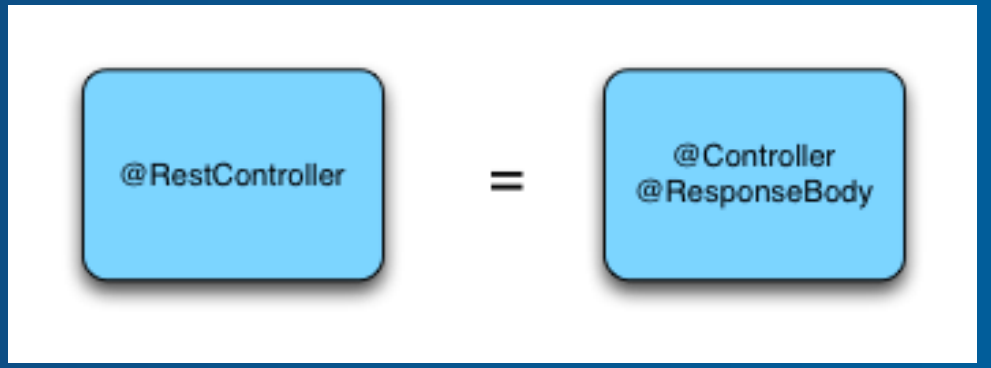
Spring Boot

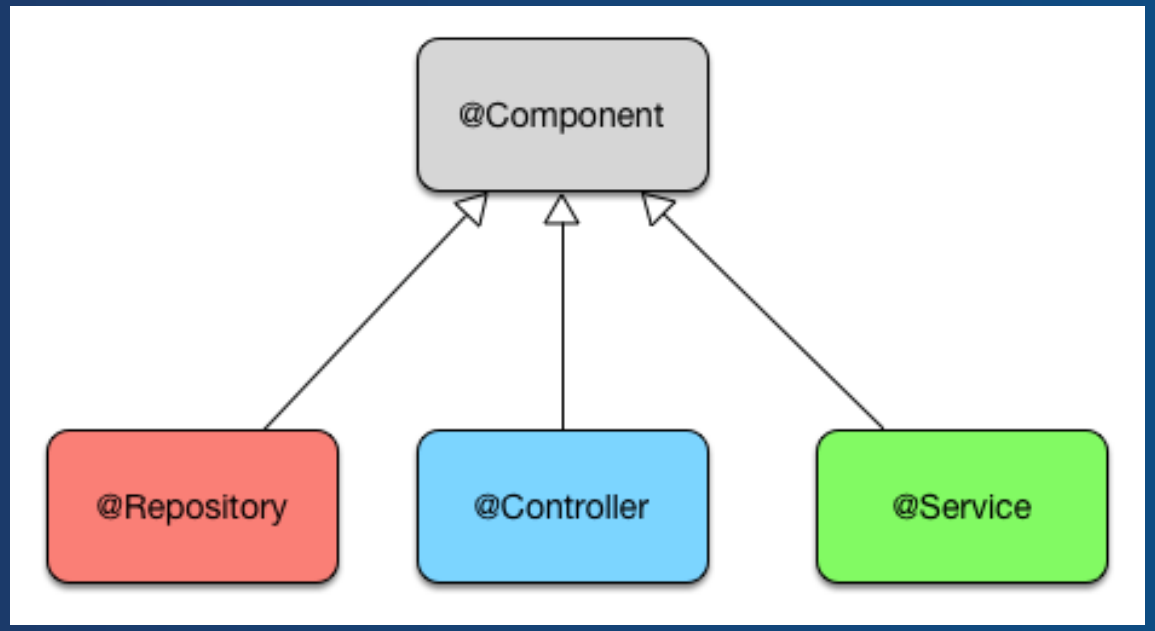
- Es un framework que permite simplificar la forma en la que se trabaja con Spring Framework y la manera en como se despliegan las aplicaciones.

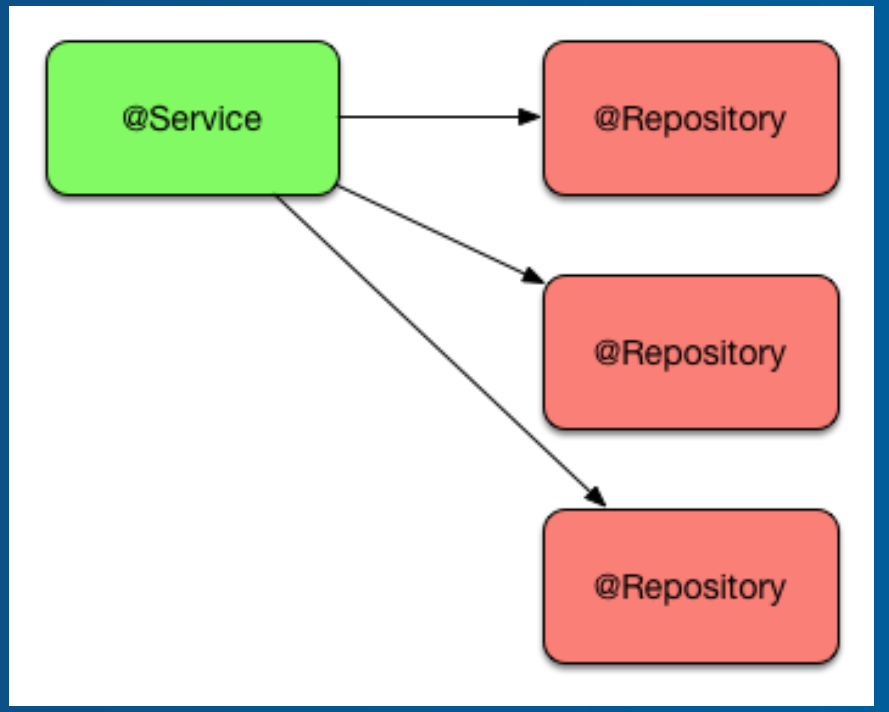


Spring Web

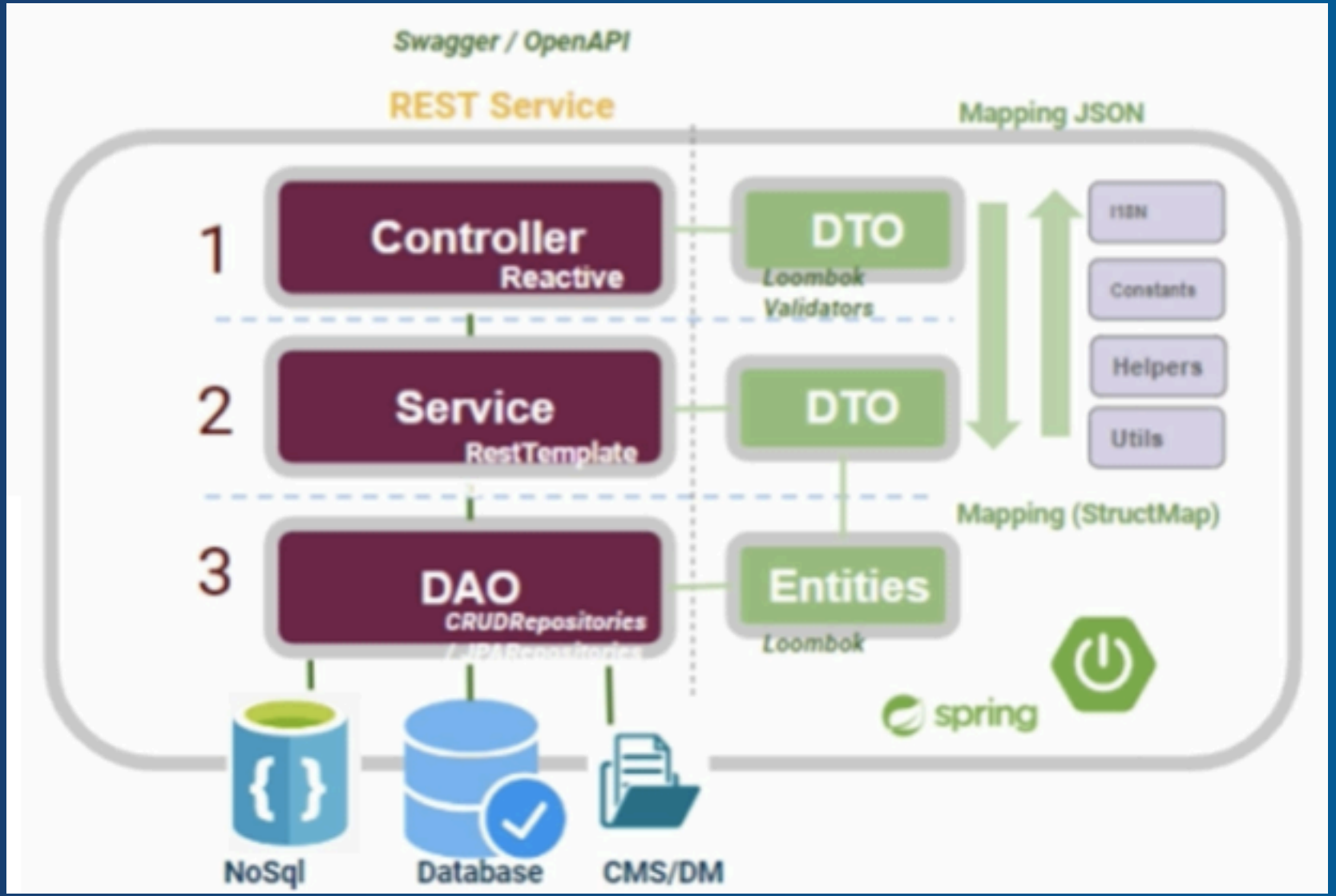
- Estereotipos





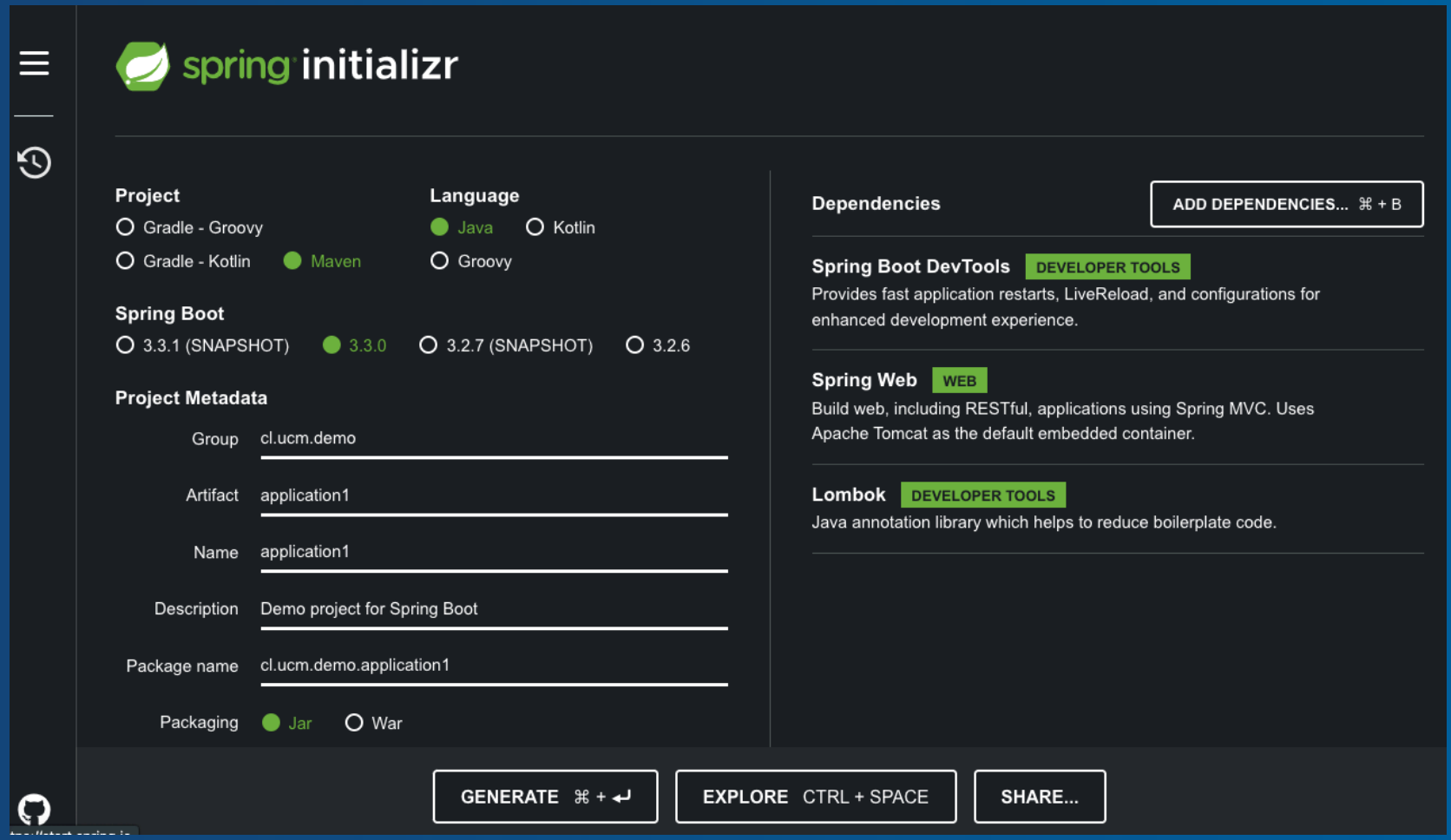


Arquitectura



Proyecto

- https://start.spring.io/



Lombok

```java
@AllArgsConstructor
public class User {
    private Integer id;
    private String name;
}
```

=

```java
public class User {
    private Integer id;
    private String name;

    public User(Integer id, String name) {
        this.id = id;
        this.name = name;
    }
}
```

Lombok

```java
@NoArgsConstructor
public class User {
    private Integer id;
    private String name;
}
```

=

```java
public class User {
    private Integer id;
    private String name;

    public User() {
    }
}
```

Lombok

```java
@Getter
@Setter
public class User {
    private Integer id;
    private String name;
    private String address;
}
```

=

```java
public class User {
    private Integer id;
    private String name;
    private String address;

    public Integer getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }

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

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }
}
```

Controllers

```
@RestController
public class DemoController {

}
```

```
@GetMapping("/hello")
public ResponseEntity<String> hello(){
    return ResponseEntity.ok("Hello Spring Boot");
}
```

PathVariable / RequestParam

Controller			HTTP Sample Request
Annotation	생략 가능	Sample Code	

@PathVariable	X	`@GetMapping("/star/{name}/age/{age}")`
		`public String hello(`
		`@PathVariable String name,`
		`@PathVariable int age`
		`){ ... }`
			GET http://localhost:8080/hello/request/star/Robbie/age/95

@RequestParam	O	`@GetMapping("/form/param")`
		`public String hello(`
		`@RequestParam String name,`
		`@RequestParam int age`
		`){ ... }`
			GET http://localhost:8080/hello/request/form/param?
			name=Robbie&age=95

	O	`@PostMapping("/form/param")`
		`public String hello(`
		`@RequestParam String name,`
		`@RequestParam int age`
		`){ ... }`
			POST
			Header
			Content type: application/x-www-form-urlencoded
			Body
			name=Robbie&age=95

PathVariable / RequestParam

```java
@GetMapping("hello2")
public ResponseEntity<Map> hello2(@RequestParam(defaultValue = "") String name){
    Map<String, String> map = new HashMap<>();
    if(!name.isEmpty()){
        map.put("message", "Hello "+name);
        return ResponseEntity.ok(map);
    }
    return ResponseEntity.noContent().build();
}
```

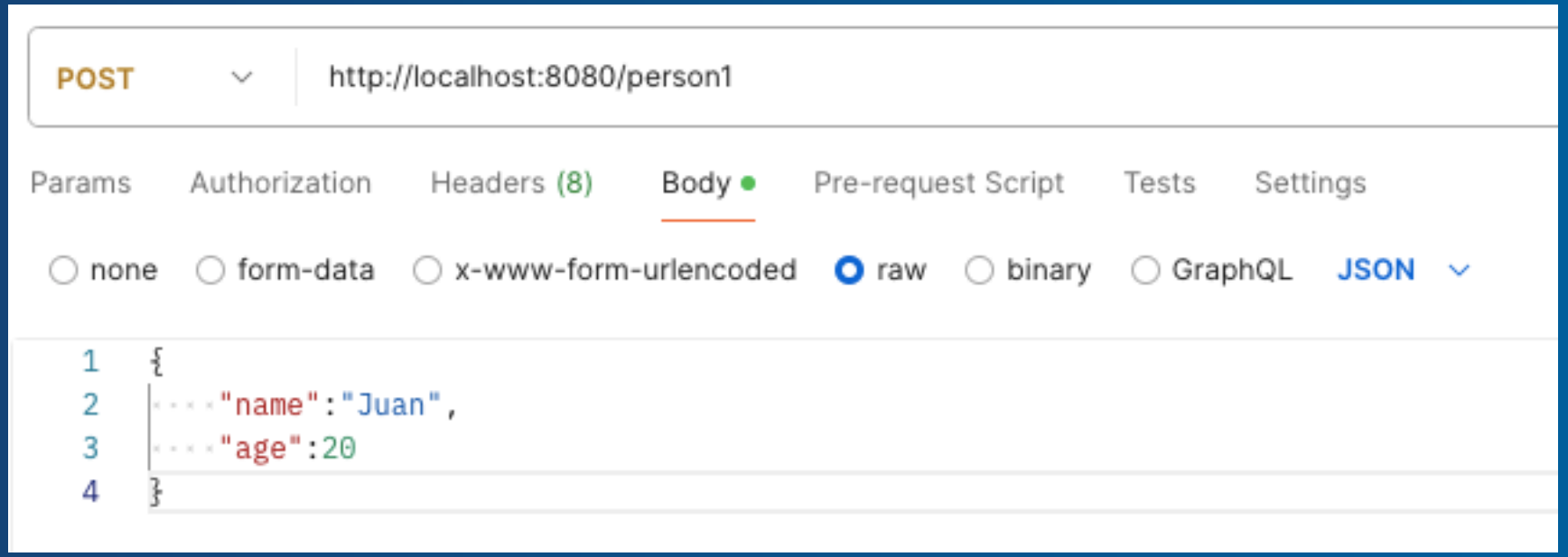
http://localhost:8080/hello2?name=Juan

http://localhost:8080/hello3/Pepe

```java
@GetMapping("/hello3/{name}")
public ResponseEntity<Map> hello3(@PathVariable String name){
    Map<String, String> map = new HashMap<>();
    if(!name.isEmpty()){
        map.put("message", "Hello "+name);
        return ResponseEntity.ok(map);
    }
    return ResponseEntity.noContent().build();
}
```

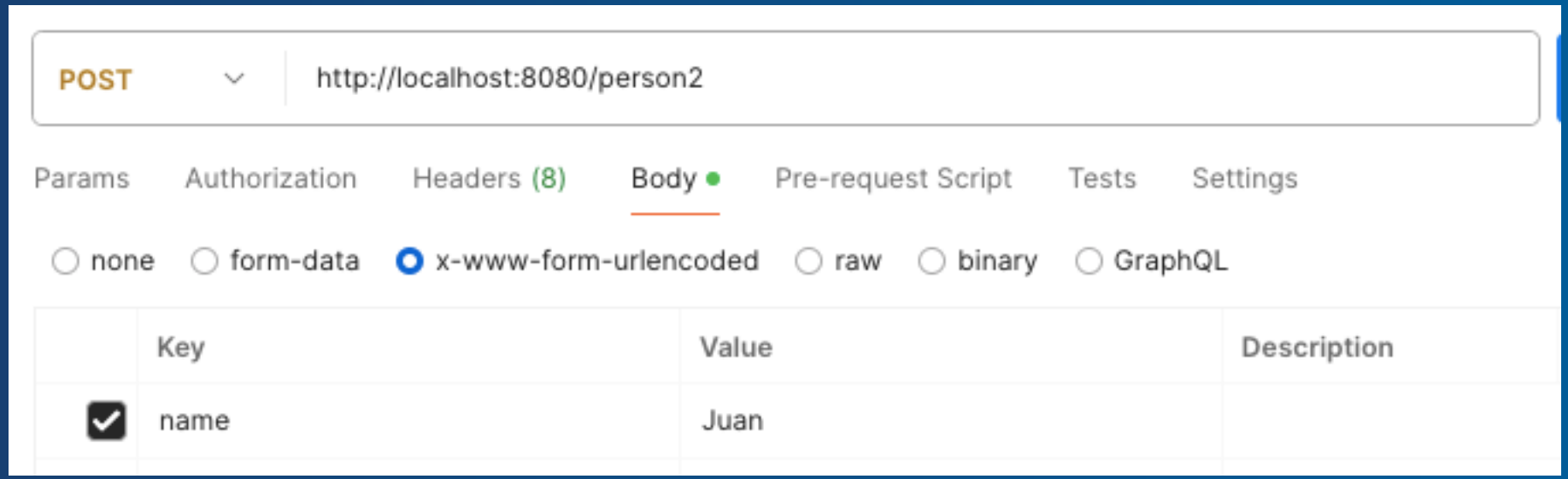
RequestBody

```java
@PostMapping("/person1")
public ResponseEntity<Person> procesa1(@RequestBody(required = true) Person person){
    person.setName(person.getName().toUpperCase());
    return ResponseEntity.ok(person);
}
```

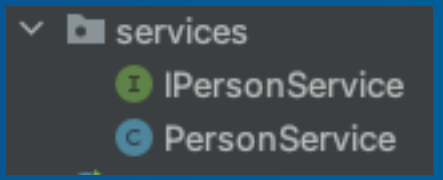


RequestParam

```java
@PostMapping("/person2")
public ResponseEntity<Map> procesa2(@RequestParam String name){
    Map<String, String> map = new HashMap<>();
    map.put("message", "Hello "+name);
    return ResponseEntity.ok(map);
}
```



Services



- Se encarga de la lógica de negocio
- Se define por medio de la notación @Service
- Se debe definir un contrato por medio de una interfaz (clean)
- Se instancia mediante la notación @Autowired

Services

```java
public interface IPersonService {

    String validarPerson(Person person);

    List<Person> getPerson();
}
```

```java
@RestController
public class DemoController {

    @Autowired
    private IPersonService service;
```

```java
@Service
public class PersonService implements IPersonService{
    @Override
    public String validarPerson(Person person) {
        if(person.getAge()<0 || person.getAge()>120){
            return "Age invalid";
        }
        if(person.getName().isEmpty()){
            return "name empty";
        }
        return "Person ok!";
    }

    @Override
    public List<Person> getPerson() {
        List<Person> list = new ArrayList<>();
        list.add(new Person( name: "Juan",  age: 20));
        list.add(new Person( name: "Pedro",  age: 21));
        list.add(new Person( name: "Pablo",  age: 23));
        list.add(new Person( name: "Maria",  age: 22));
        return list;
    }
}
```

Repository Entity

- Para trabaja con base de datos depende del motor a usar para instalar las dependencias

```
implementation 'org.springframework.boot:spring-boot-starter-data-jpa:3.2.4'
runtimeOnly 'com.h2database:h2'
```

- H2 es una base de datos local que trae springboot

Repository Entity

- Es necesario definir las siguientes propiedades de base de datos

```
spring.h2.console.enabled=true
spring.datasource.url=jdbc:h2:mem:dmallapp
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

Repository Entity

- Es necesario definir las siguientes propiedades de base de datos

```
spring.h2.console.enabled=true
spring.datasource.url=jdbc:h2:mem:dmallapp
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

Mysql

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/coffee?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=root

spring.jpa.hibernate.ddl-auto=update
```

- Gradle

```
runtimeOnly 'com.mysql:mysql-connector-j'
```

Repository Entity

- Desde Java se puede crear la tabla de la base de datos o bien, se puede usar una ya existente.

```java
@Entity
@Table(name = "locals")
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class Local {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String name;
    private String floor;
    private String code;
    @Lob
    String fotoBase64;
}
```

Repository Entity

- Para definir la interfaz repository se utiliza la clase JPARepository, la cual hereda métodos ya diseñados para un CRUD

```
public interface LocalRepository extends JpaRepository<Local, Long> {
}
```

- Para programar los métodos de esta interfaz se debe contar con un servicio junto con su contrato

```
@Service
public class LocalServiceImpl implements LocalService{
    @Autowired
    private LocalRepository localRepository;
```

Relaciones

- Foreign key, id_category_fk es una clave foránea que esta asociada a una entidad category con clave primaria llamada id_category

```
@ManyToOne(fetch = FetchType.EAGER)
@JoinColumn(name = "id_category_fk", referencedColumnName = "id_category", updatable = false,insertable = false )
private CategoryEntity category;
```

QueryMethods

- @Quey(value="SQL", nativeQuery="true")