

UCM

Programación de Aplicaciones Web **Electivo III**

Román Gajardo



<https://github.com/romanncodes>



roman.gajardo@gmail.com



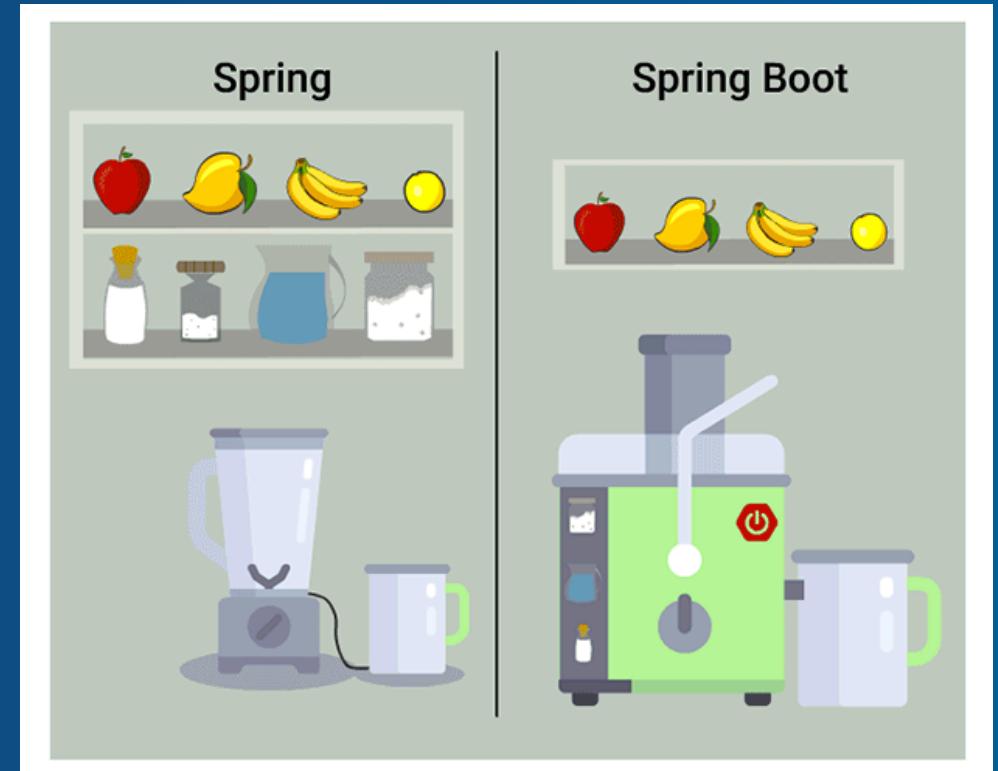


Contenidos

- Java
- Spring Boot

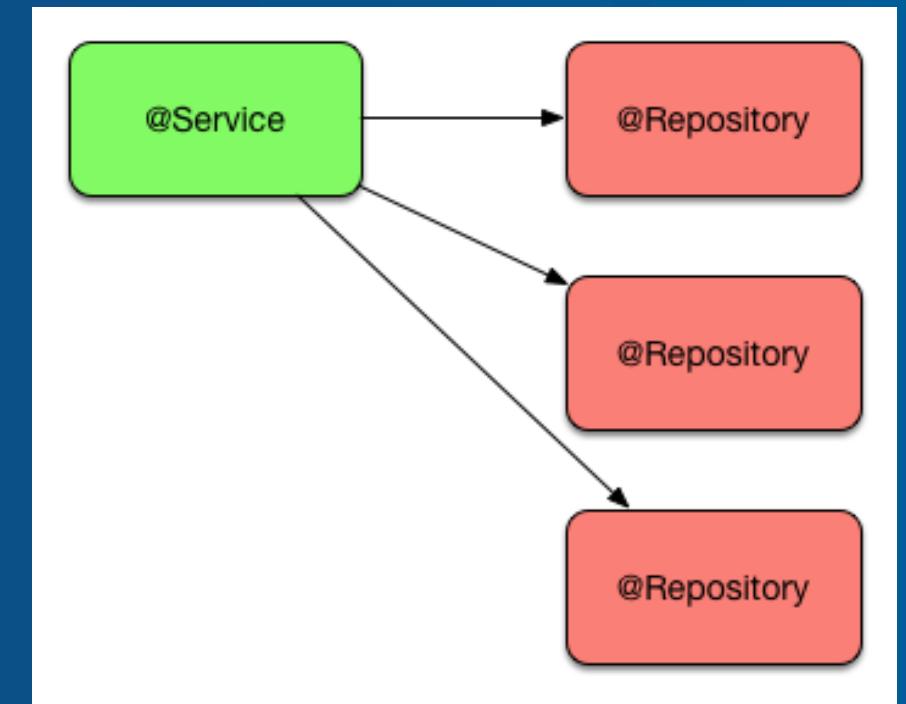
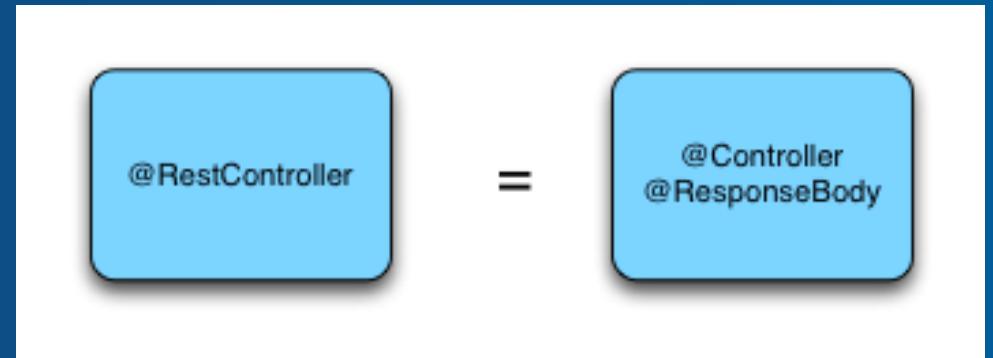
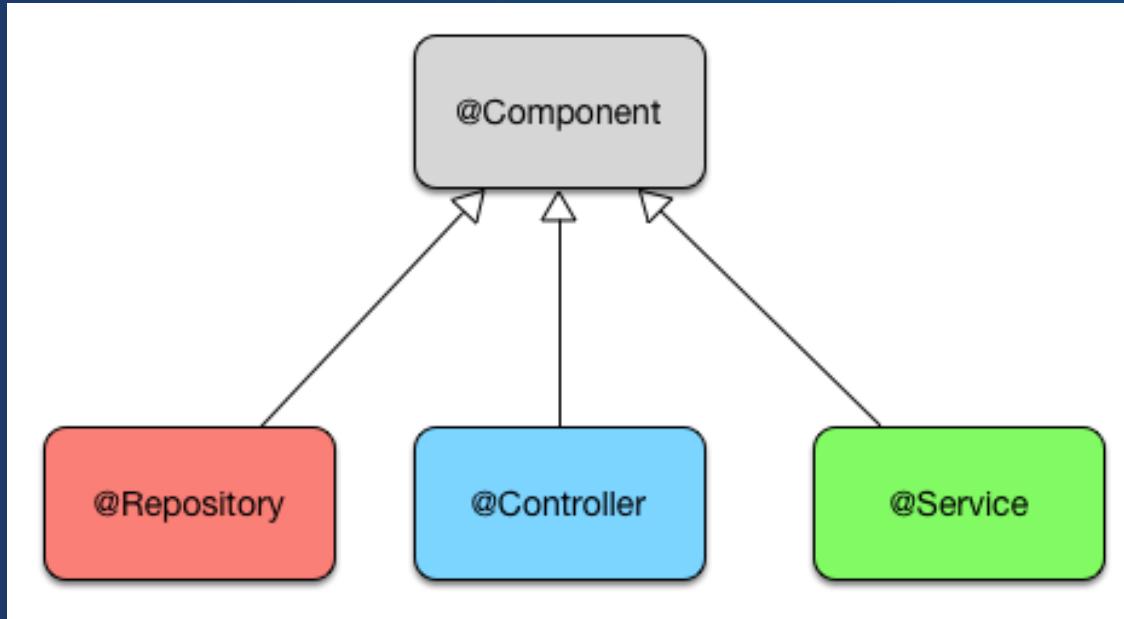
Spring Boot

- 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.

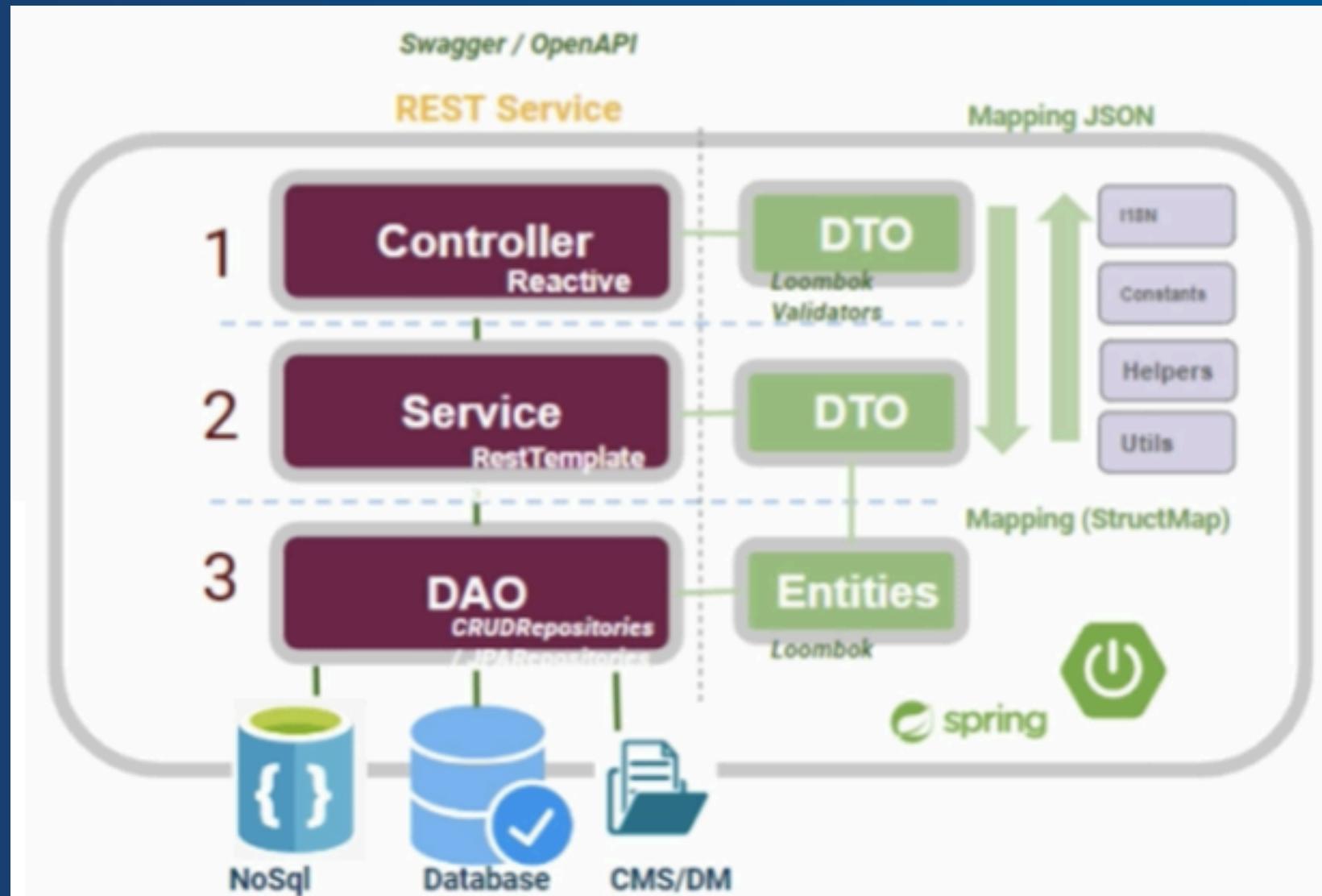


Spring Web

- Estereotipos



Arquitectura



Proyecto

- <https://start.spring.io/>

The screenshot shows the Spring Initializr web application interface. At the top, there's a navigation bar with a menu icon, a refresh icon, and the "spring initializr" logo. Below the header, there are sections for "Project", "Language", "Spring Boot", and "Dependencies".

Project: Options include Gradle - Groovy, Gradle - Kotlin, Maven (selected), and Groovy.

Language: Options include Java (selected) and Kotlin.

Spring Boot: Options include 3.3.1 (SNAPSHOT), 3.3.0 (selected), 3.2.7 (SNAPSHOT), and 3.2.6.

Project Metadata fields:

- Group: cl.ucm.demo
- Artifact: application1
- Name: application1
- Description: Demo project for Spring Boot
- Package name: cl.ucm.demo.application1
- Packaging: Jar (selected)

Dependencies section:

- Spring Boot DevTools** (DEVELOPER TOOLS): Provides fast application restarts, LiveReload, and configurations for enhanced development experience.
- Spring Web** (WEB): Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
- Lombok** (DEVELOPER TOOLS): Java annotation library which helps to reduce boilerplate code.

At the bottom, there are three buttons: "GENERATE" (⌘ + ↩), "EXPLORE" (CTRL + SPACE), and "SHARE...".

Lombok

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



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

Lombok

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



```
public class User {  
    private Integer id;  
    private String name;  
}  
public User() {  
}
```

Lombok

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



```
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	<pre>@GetMapping("/star/{name}/age/{age}") public String hello(@PathVariable String name, @PathVariable int age) { ... }</pre>	GET http://localhost:8080/hello/request/star/Robbie/age/95
@RequestParam	<input type="radio"/>	<pre>@GetMapping("/form/param") public String hello(@RequestParam String name, @RequestParam int age) { ... }</pre>	GET http://localhost:8080/hello/request/form/param? name=Robbie&age=95
	<input type="radio"/>	<pre>@PostMapping("/form/param") public String hello(@RequestParam String name, @RequestParam int age) { ... }</pre>	POST Header Content type: application/x-www-form-urlencoded Body name=Robbie&age=95

PathVariable / RequestParam

```
@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

```
@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

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

The screenshot shows the Postman application interface. At the top, it displays a POST method and the URL `http://localhost:8080/person1`. Below the URL, there are tabs for Params, Authorization, Headers (8), Body (selected), Pre-request Script, Tests, and Settings. Under the Body tab, the content type is set to raw JSON. The JSON payload is defined as follows:

```
1 {
2   "name": "Juan",
3   "age": 20
4 }
```

RequestParam

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

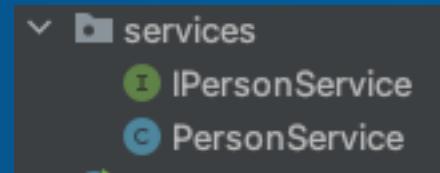
POST http://localhost:8080/person2

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

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

	Key	Value	Description
<input checked="" type="checkbox"/>	name	Juan	

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

```
public interface IPersonService {  
  
    String validarPerson(Person person);  
  
    List<Person> getPerson();  
}
```

```
@RestController  
public class DemoController {  
  
    @Autowired  
    private IPersonService service;
```

```
@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 trabajar 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.

```
@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

- `@Query(value="SQL", nativeQuery="true")`