

## Tabla de Contenidos



- [Spring Boot REST JPA y Relaciones](#)
- [@JsonIgnore y sus limitaciones](#)
- [Spring Boot REST y JPA](#)
- [Otros artículos relacionados](#)

# CURSO SPRING REST GRATIS APUNTATE!!

Spring Boot REST JPA . ¿Como podemos encajar de una forma natural JPA y Domain Driven Design con las arquitecturas REST que manejamos habitualmente . Este es uno de los dilemas a los que todos nos enfrentamos en el día a día . Existen muchas soluciones algunas son muy muy sencillas como el uso de DTOs pero en muchas ocasiones nos encontramos con estructuras algo más complejas en las que nos comienzan a aparecer las dudas . Un ejemplo claro de este tipo de situaciones son las relaciones a nivel de JPA . Imaginemonos que disponemos de dos clases que están relacionadas Factura y Linea de Factura . Un ejemplo a nivel de código podría ser algo como :

```
package com.arquitecturajava.rest;
```

```
import java.util.List;
```

```
import javax.persistence.Entity;
```

```
import javax.persistence.Id;
```

```
import javax.persistence.OneToMany;
```

```
import javax.persistence.Table;
```

```
import com.fasterxml.jackson.annotation.JsonIgnore;

@Entity
@Table(name="facturas")
public class Factura {

    @Id
    private int numero;
    private String concepto;
    @OneToMany(mappedBy="factura")
    private List<LineaFactura> lineas;
    public List<LineaFactura> getLineas() {
        return lineas;
    }
    public void setLineas(List<LineaFactura> lineas) {
        this.lineas = lineas;
    }
    public int getNumero() {
        return numero;
    }
    public void setNumero(int numero) {
        this.numero = numero;
    }
    public String getConcepto() {
        return concepto;
    }
    public void setConcepto(String concepto) {
        this.concepto = concepto;
    }
    public Factura() {
        super();
    }
}
```

```
}  
public Factura(int numero, String concepto) {  
    super();  
    this.numero = numero;  
    this.concepto = concepto;  
}  
public Factura(int numero) {  
    super();  
    this.numero = numero;  
}  
@Override  
public int hashCode() {  
    final int prime = 31;  
    int result = 1;  
    result = prime * result + numero;  
    return result;  
}  
@Override  
public boolean equals(Object obj) {  
    if (this == obj)  
        return true;  
    if (obj == null)  
        return false;  
    if (getClass() != obj.getClass())  
        return false;  
    Factura other = (Factura) obj;  
    if (numero != other.numero)  
        return false;  
    return true;  
}  
}
```

```
package com.arquitecturajava.rest;

import javax.persistence.EmbeddedId;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.MapId;
import javax.persistence.Table;

@Entity
@Table(name="lineasfacturas")
public class LineaFactura {
    @EmbeddedId
    private LineaFacturaPK lineaPK;
    private double importe;
    @MapId("factura_numero")
    @ManyToOne()
    @JoinColumn()
    private Factura factura;
    public double getImporte() {
        return importe;
    }
    public void setImporte(double importe) {
        this.importe = importe;
    }
    public Factura getFactura() {
        return factura;
    }
    public void setFactura(Factura factura) {
        this.factura = factura;
    }
}
```

```
}  
public int getFactura_numero() {  
    return lineaPK.getFactura_numero();  
}  
public void setFactura_numero(int factura_numero) {  
    lineaPK.setFactura_numero(factura_numero);  
}  
public int getNumero() {  
    return lineaPK.getNumero();  
}  
public void setNumero(int numero) {  
    lineaPK.setNumero(numero);  
}  
public LineaFactura(int numero, int numeroFactura) {  
    this.lineaPK= new LineaFacturaPK();  
    this.lineaPK.setNumero(numero);  
    this.lineaPK.setFactura_numero(numeroFactura);  
}  
public LineaFactura() {  
    super();  
}  
@Override  
public int hashCode() {  
    final int prime = 31;  
    int result = 1;  
    result = prime * result + ((lineaPK == null) ? 0 :  
lineaPK.hashCode());  
    return result;  
}  
@Override  
public boolean equals(Object obj) {
```

```
if (this == obj)
    return true;
if (obj == null)
    return false;
if (getClass() != obj.getClass())
    return false;
LineaFactura other = (LineaFactura) obj;
if (lineaPK == null) {
    if (other.lineaPK != null)
        return false;
} else if (!lineaPK.equals(other.lineaPK))
    return false;
return true;
}
}
```

Disponemos de dos clases relacionadas entre sí en una relación de 1 a n

Queremos construir un servicio REST que nos devuelve la lista de facturas . En principio todo es muy sencillo basta con usar Spring Boot e inicializarlo con los starters correspondientes Spring Web , JPA etc.

**TODOS LOS CURSOS  
PROFESIONALES  
25\$/MES  
APUNTATE!!**

<dependency>

```

    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>

```

Una vez disponemos de los starters podemos diseñar un repositorio de JPA que se encargue de devolver una lista de Facturas al cliente :

```

package com.arquitecturajava.rest;

import java.util.ArrayList;
import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.TypedQuery;
import javax.persistence.criteria.CriteriaBuilder;

```

```
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.ParameterExpression;
import javax.persistence.criteria.Predicate;
import javax.persistence.criteria.Root;

import org.springframework.stereotype.Repository;

@Repository
public class FacturaRepository {

    @PersistenceContext
    private EntityManager em;

    public List<Factura> buscarTodas() {

        return em.createQuery("select f from Factura f",
Factura.class).getResultList();
    }

}
```

Recordemos que para que la parte de persistencia de JPA funcione necesitamos integrar la conexión a través del aplicación properties:

```
spring.datasource.url=jdbc:mysql://localhost:8889/rest2
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver.class=com.mysql.cj.jdbc.Driver
spring.jpa.properties.hibernate.dialect =
org.hibernate.dialect.MySQL5Dialect
```



Hecho esto es momento de diseñar un servicio REST que nos devuelva la lista de facturas:

```
package com.arquitecturajava.rest;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/facturas")
public class FacturaController {

    @Autowired
    private FacturaRepository repositorio;
    @GetMapping
    public List<Factura> buscarTodas() {
        return repositorio.buscarTodas();
    }
}
```

## Spring Boot REST JPA y Relaciones

En principio con esta configuración será suficiente para obtener el listado de Facturas sin embargo si la inicializamos nos encontraremos con el siguiente problema:



permitirá eliminar de las estructuras JSON las relaciones.

```
@OneToMany(mappedBy="factura")
@JsonIgnore
private List<LineaFactura> lineas;
```

De esta manera no entraremos en una situación recursiva y se mostrarán los datos que existen en la tabla:

```
[{"numero":1,"concepto":"tablet"}, {"numero":2,"concepto":"televisor"}, {"numero":3,"concepto":"auricular"}, {"numero":4,"concepto":"tablet"}, {"numero":5,"concepto":"telefono"}, {"numero":6,"concepto":"ordenador"}]
```

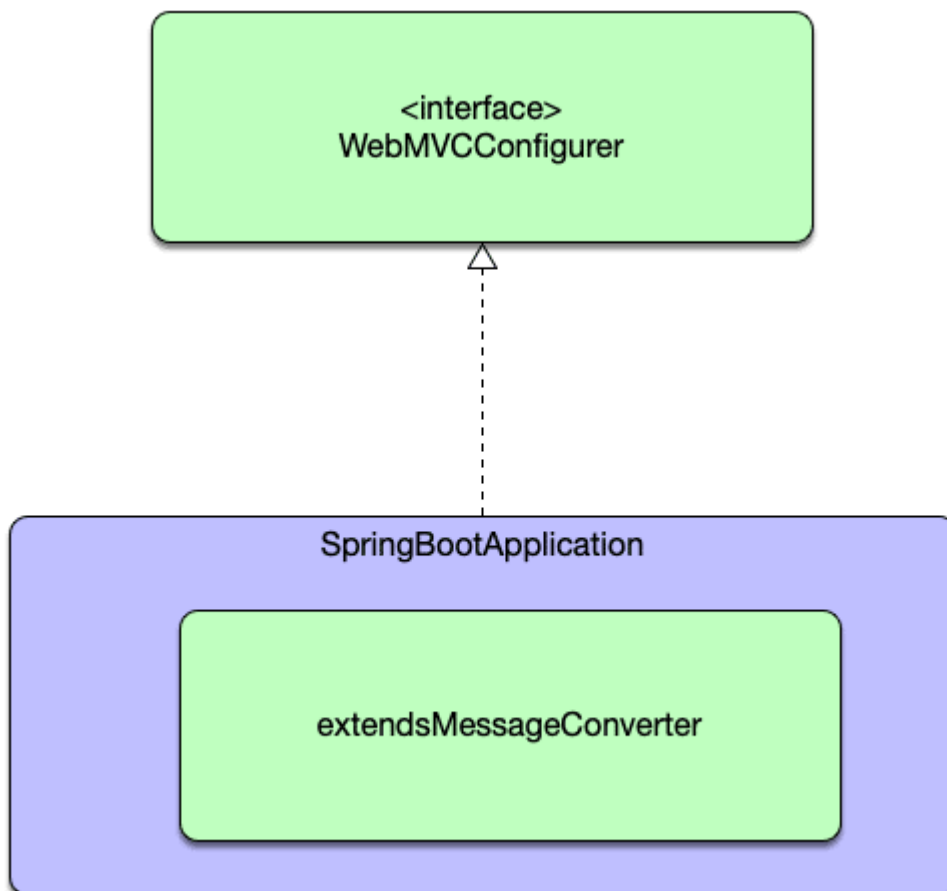
## @JsonIgnore y sus limitaciones

Aunque esto nos permite solventar el problema .Hay situaciones con @JsonIgnore que se convierten en problemáticas ya que por ejemplo tenemos que activar las anotaciones en todas la relaciones existentes . Otra solución más elegante es configurar JackSon que es la librería que se encarga de transformar los objetos Java en JSON y añadirla un modulo adicional concretamente un módulo para Hibernate 5 que manejo de objetos de persistencia.

```
<dependency>
  <groupId>com.fasterxml.jackson.datatype</groupId>
  <artifactId>jackson-datatype-hibernate5</artifactId>
</dependency>
```

## Spring Boot REST y JPA

Una vez instalado este módulo sera el propio Jackson el que se encargará de la gestión de las relaciones de Hibernate a nivel de servicios REST. Eso sí habrá que solicitar registrar el módulo a nivel de SpringBoot.



De esta manera personalizamos la configuración de SpringBoot y Añadimos una extension para conversión de mensajería Http.

```

@SpringBootApplication
public class RestApplication implements WebMvcConfigurer {

    public static void main(String[] args) {
        SpringApplication.run(RestApplication.class, args);
    }

    @Override
    public void
    extendMessageConverters(List<HttpMessageConverter<?>> converters) {

        for (HttpMessageConverter<?> converter : converters) {
  
```

```
        if (converter instanceof
org.springframework.http.converter.json.MappingJackson2HttpMessageConv
erter) {
            ObjectMapper mapper =
((MappingJackson2HttpMessageConverter) converter).getObjectMapper();
            mapper.registerModule(new Hibernate5Module());
        }
    }
}
}
```

Esto hará que los resultado salgan mucho mas directos sin relaciones :

```
[{"numero":1,"concepto":"tablet","lineas":null},{"numero":2,"concepto":"televisor","lineas":null}
{"numero":4,"concepto":"tablet","lineas":null},{"numero":5,"concepto":"telefono","lineas":null},{
```

De esta forma tendremos configurado Spring Boot REST JPA sin tener que apoyarnos en @JsonIgnore para las operaciones fundamentales de consultas a la base de datos y construcciones de gráficos.

## Otros artículos relacionados

**CURSO SPRING REST  
GRATIS  
APUNTATE!!**

1. [Spring Boot JDBC y su configuración](#)
2. [Spring Boot Load Data y testing](#)
3. [Curso de Spring Boot Gratis en Español](#)

#### 4. Spring Batch Hello World y su configuración