

## **Índice de contenido**

# SPRING BOOT

## 1. CONFIGURACIÓN

### 1.1. Creación del proyecto:

1.1.1. Creación del proyecto mediante la web de spring boot 'https://start.spring.io/'.

1.1.2. Creación del proyecto mediante ecclipse: new/proyect/New Spring Starter Project

### 1.2. Configuración

Generate a Maven Project with Java and Spring Boot 2.0.5

#### Project Metadata

Artifact coordinates

Group

Artifact

#### Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

Selected Dependencies

JPA MySQL Web

1.2.1. Group: Ruta inicial de las clases del proyecto.

1.2.2. Artifact: Nombre del fichero de compilación del proyecto.

1.2.3. Dependencies: Podemos introducir las dependencias que queremos introducir en el proyecto.

- Fichero pom.xml

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
  <java.version>1.8</java.version>
  <jackson.version>2.8.10</jackson.version>
  <jaxb-api.version>2.2.11</jaxb-api.version>
</properties>

<dependencies>

  <!-- Driver de conexión mysql -->
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.11</version>
  </dependency>
  <!-- Convierte una clase a Json y Json a una clase -->
  <dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>${jackson.version}</version>
  </dependency>
  <!-- Convierte una clase a Xml y Xml a una clase -->
  <dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>${jaxb-api.version}</version>
    <scope>runtime</scope>
  </dependency>

</dependencies>
```

- Fichero de configuración del proyecto 'properties'.

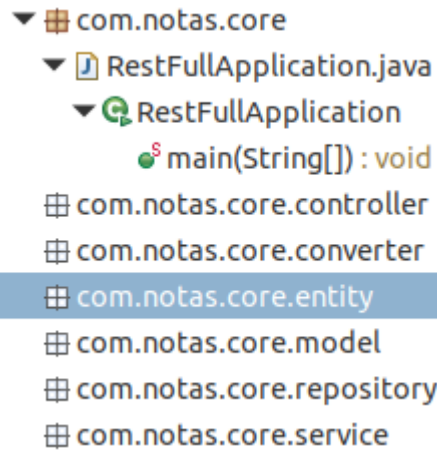
# SPRING BOOT

```
#Puerto donde corre el servidor
server.port=8080
#Muestra el error 404
server.error.whitelabel.enabled=false

#Datos de conexión con la base de datos
spring.datasource.url=jdbc:mysql://localhost:3306/restSpring?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC&useSSL=false
spring.datasource.username=root
spring.datasource.password=Icaro1979++

#Muestra el código sql por la consola
spring.jpa.show-sql=true
#Cuando se corre el proyecto se actualiza/crea/ la Base de Datos con los cambios
spring.jpa.hibernate.ddl-auto=update
#Estrategia de la base de datos
spring.jpa.hibernate.naming-strategy=org.hibernate.cfg.ImprovedNamingStrategy
#Indica que tipo de base de datos vamos ha usar.
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
```

- Estructura de ficheros.



```
▼ com.notas.core
  ▼ RestFullApplication.java
    ▼ RestFullApplication
      main(String[]) : void
  com.notas.core.controller
  com.notas.core.converter
  com.notas.core.entity
  com.notas.core.model
  com.notas.core.repository
  com.notas.core.service
```

- Controller:
- Converter:
- Entity:
- Model:
- Repository:
- Service:

# SPRING BOOT

## 2. ENTIDADES Y MODELOS.

- Entity.

```
import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

/**
 * @author fmgar
 *
 */
@Entity
@Table(name="NOTA")
public class Nota implements Serializable{

    /**
     *
     */
    private static final long serialVersionUID = 1165474281158866341L;

    @Id
    @GeneratedValue
    @Column(name="ID")
    private Long id;

    @Column(name="NOMBRE", unique=true)
    private String nombre;

    @Column(name="TITULO")
    private String titulo;

    @Column(name="DESCRIPCION")
    private String contenido;

    public Nota() {

    }

    public Nota(Long id, String nombre, String titulo, String contenido) {
        this.id = id;
        this.nombre = nombre;
        this.titulo = titulo;
        this.contenido = contenido;
    }
}
```

- Permite mapear la Clase con una tabla de la base de datos por medio de anotaciones de JPA.
  - @Entity: Definimos que la clase es una entidad.
  - @Table: Definimos que la clase mapea una tabla, identificamos la tabla que mapea la clase por medio del nombre.
  - @Id: Define que la propiedad anotada representa al campo identificador de la tabla.
  - @GeneratedValue: Define como se va a generar el valor de la propiedad identificadora de la tabla
  - @Column: Define que la propiedad anotada representa un campo de la tabla, pudiendo definir el nombre de la tabla, el tipo de dato que va a guardar, si es un campo que no se van a repetir los valores (unique), etc.

## SPRING BOOT

- Si utilizamos la anotación `@XmlRootElement` antes de la definición de la clase indicamos que vamos a utilizar xml en vez e json para convertir los datos a la clase entidad.
- Se va a trabajar con el por medio de los repositorios.

- Model.

```
import com.notas.core.entity.Nota;

public class MNota {

    private Long id;

    private String nombre;

    private String titulo;

    private String contenido;

    public MNota() {

    }

    public MNota(Long id, String nombre, String titulo, String contenido) {
        this.id = id;
        this.nombre = nombre;
        this.titulo = titulo;
        this.contenido = contenido;
    }

    public MNota(Nota nota) {
        this.id = nota.getId();
        this.nombre = nota.getNombre();
        this.titulo = nota.getTitulo();
        this.contenido = nota.getContenido();
    }

    public Long getId() {
        return id;
    }

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

- Representa la entidad como el modelo de dato que vamos a devolver al usuario en la vista.
- Debamos crear un constructor que construya el modelo a partir de un objeto de la entidad que representa.
- Se va a trabajar con el por medio de los controladores.

# SPRING BOOT

## 3. REPOSITORIOS Y CONVERTIDORES.

- Repositorio.

```
package com.notas.core.repository;

import java.io.Serializable;
import java.util.List;

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

import com.notas.core.entity.Nota;

@Repository("repositorio")
public interface NotaRepository extends JpaRepository<Nota, Serializable>{

    public abstract Nota findByNombre(String nombre);

    public abstract List<Nota> findByTitulo(String titulo);

    public abstract Nota findByNombreAndTitulo(String nombre, String titulo);
}
```

- Se encarga de hacer peticiones a la base de datos y actualizar los datos mediante entidades relacionadas con las tablas a consultar o actualizar.
- Se crea mediante una interfaz que extiende de `JpaRepository`, debemos definir la entidad que vamos a utilizar.
- **@Repository**: Mediante esta anotación definimos que la interfaz es un repositorio y el nombre del bean para inyectar el repositorio en el servicio.
- Podemos definir métodos abstractos que van a ser los encargados de realizar las consultas a la base de datos y devolvernos una entidad o una lista de entidades rellenas con el resultado de la consulta.
  - `NombreEntidad findByNombreCampo`: Devuelve una entidad que el valor del campo en la tabla que tiene el nombre del método coincida con el parámetro que le pasamos al método.
  - `List<NombreEntidad> findByNombreCampo`: Devuelve una lista de entidades en las que el valor del campo en la tabla que tiene el nombre del método coincida con el parámetro que le pasamos al método.
  - `NombreEntidad findByNombreCampo1AndNombreCampo2`: Devuelve una entidad realizando una consulta en la que coinciden los valores de los parámetros en los campos definidos en el nombre del método.

- Convertidor

## SPRING BOOT

```
package com.notas.core.converter;

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

import org.springframework.stereotype.Component;

import com.notas.core.entity.Nota;
import com.notas.core.model.MNota;

@Component("convertidor")
public class Convertidor {
    public List<MNota> convertirLista(List<Nota> notas){
        List<MNota> mnotas = new ArrayList<MNota>();
        for(Nota nota: notas) {
            mnotas.add(new MNota(nota));
        }
        return mnotas;
    }
}
```

- **@Component**: Mediante esta anotación definimos que la clase es un componente y el nombre del bean para inyectar el componente en el servicio.
- Por medio del método convertirLista convierte una lista de tipo de entidad a un tipo de modelo para devolvérselo al Usuario en la vista.

# SPRING BOOT

## 4. SERVICIO

```
package com.notas.core.service;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Service;
import com.notas.core.converter.Convertidor;
import com.notas.core.entity.Nota;
import com.notas.core.model.MNota;
import com.notas.core.repository.NotaRepository;

@Service("servicio")
public class NotaService {
    @Autowired
    @Qualifier("repositorio")
    private NotaRepository repositorio;

    @Autowired
    @Qualifier("convertidor")
    private Convertidor convertidor;

    public boolean actualizar(Nota nota) {
        try {
            repositorio.save(nota);
            return true;
        } catch (Exception ex) {
            return false;
        }
    }

    public boolean borrar(String nombre, Long id) {
        try {
            Nota nota = repositorio.findByNombreAndId(nombre, id);
            repositorio.delete(nota);
            return true;
        } catch (Exception ex) {
            return false;
        }
    }

    public List<MNota> obtener() {
        return convertidor.convertirLista(repositorio.findAll());
    }

    public MNota obtenerPorNombreYTitulo(String nombre, String titulo) {
        return new MNota(repositorio.findByNombreAndTitulo(nombre, titulo));
    }
}
```

- El Servicio define una serie de métodos para interactuar con los repositorios de la aplicación, pudiendo consultar y actualizar datos en las tablas de la base de datos, etc.
- **@Service**: La anotación indica que la clase es un servicio y define el nombre del bean para inyectar el servicio en el controlador.
- **@Autowired**: La anotación indicamos a spring que vamos a inyectar un bean
- **@Qualifier**: Con la anotación indicamos el nombre del bean a inyectar.



# SPRING BOOT

## 5. CONTROLADOR

### 5.1. Creación de controlador.

```
package com.notas.core.controller;

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

import com.notas.core.service.NotaService;

@RestController
@RequestMapping("/v1")
public class NotaController {
    @Autowired
    @Qualifier("servicio")
    private NotaService servicio;
}
```

- **@RestController**: Esta anotación indica que esta clase es la controladora de una Api Rest.
- **@RequestMapping**: Esta anotación mapea el controlador, también indicamos la versión del Api Rest.
- **@Autowired** y **@Qualifier**: Por medio de estas anotaciones inyectamos las dependencias que necesitamos de los servicios.

### 5.2. Método crear.

```
@PutMapping("/nota")
public boolean agregarNota(@RequestBody @Valid Nota nota) {
    return servicio.actualizar(nota);
}
```

- **@PutMapping**: Mediante esta anotación indicamos que el método atenderá las peticiones put que nos lleguen por medio de la url /nota.
- **@RequestBody**: Esta anotación nos permite capturar los datos contenidos en el cuerpo de la petición.
- **@Valid**: Esta anotación permite convertir en json que nos llega en el cuerpo de la petición a una clase que indicamos como parámetro del método.

### 5.3. Método actualizar.

```
@PostMapping("/nota")
public boolean actualizarNota(@RequestBody @Valid Nota nota) {
    return servicio.actualizar(nota);
}
```

- **@PostMapping**: Mediante esta anotación indicamos que el método atenderá las peticiones post que nos lleguen por medio de la url /nota.
- **@RequestBody**: Esta anotación nos permite capturar los datos contenidos en el cuerpo de la petición.
- **@Valid**: Esta anotación permite convertir en json que nos llega en el cuerpo de la petición a una clase que indicamos como parámetro del método.

## SPRING BOOT

### 5.4. Método eliminar.

```
@DeleteMapping("/nota/{id}/{nombre}")
public boolean borrarNota(@PathVariable("id") Long id,
    @PathVariable("nombre") String nombre) {
    return servicio.borrar(nombre, id);
}
```

- **@DeleteMapping:** Mediante esta anotación indicamos que el método atenderá las peticiones delete que nos lleguen por medio de la url /nota. En la url le indicamos los parámetros que va a recibir en la petición entre /{ nombreParametro}
- **@PathVariable:** Mediante esta anotación recuperamos un valor de un parámetro indicando el nombre del parámetro a recuperar.