El uso de Spring Boot JPA es cada día mas importante ya que poco a poco más proyectos van pasando de usar Spring clásico a ser arrancados con Spring Boot . Por lo tanto configurar Spring Boot para trabajar con JPA es casi siempre necesario. Vamos a ver cómo hacerlo. El primer paso es generar un proyecto de Spring Boot que soporte JPA en este caso es sumamente sencillo ya que solo necesitamos un starter, vamos a ver cómo queda el fichero de maven.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>

	<groupId>com.arquitecturajava</groupId>
	<artifactId> jpa</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<packaging>jar</packaging>

	<name> jpa</name>
	<description>Demo project for Spring Boot</description>

	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>2.0.6.RELEASE</version>
		<relativePath /> <!-- lookup parent from repository -
->
	</parent>
```

```
        <properties>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncod
ing>
                <java.version>1.8</java.version>
        </properties>

        <dependencies>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-data-
jpa</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-
test</artifactId>
                        <scope>test</scope>
                </dependency>


                <!--
https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
                <dependency>
                        <groupId>mysql</groupId>
                        <artifactId>mysql-connector-java</artifactId>
                        <version>5.1.47</version>
                </dependency>

        </dependencies>
```

```
        <build>
                <plugins>
                        <plugin>
<groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-maven-
plugin</artifactId>
                        </plugin>
                </plugins>
        </build>


</project>
```

Spring Boot JPA y entidades

Una vez que tenemos configurado el proyecto de arranque el primer paso es generar una entidad que pueda ser persistida o seleccionada por JPA.

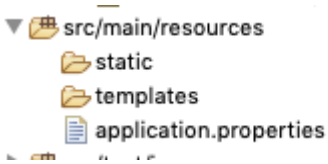
```
package com.arquitecturajava.jpa.models;

import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Persona {

        @Id
        private String nombre;
        private String apellidos;
        private int edad;
        public String getNombre() {
```

```
		return nombre;
	}
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
	public String getApellidos() {
		return apellidos;
	}
	public void setApellidos(String apellidos) {
		this.apellidos = apellidos;
	}
	public int getEdad() {
		return edad;
	}
	public void setEdad(int edad) {
		this.edad = edad;
	}
	public Persona(String nombre, String apellidos, int edad) {
		super();
		this.nombre = nombre;
		this.apellidos = apellidos;
		this.edad = edad;
	}
	public Persona() {
		super();
	}
}
```

Spring boot JPA properties

Ya tenemos la entidad definida. El siguiente paso es configurar los parámetros de

configuración contra la base de datos usando el fichero de application.properties que Spring Boot nos provee en la estructura de carpetas.



En el deberemos de rellenar las propiedades clásicas con una estructura determinada haciendo referencia a un datasource y al propio hibernate.

spring.datasource.url=jdbc:mysql://localhost:8889/springjpa

spring.datasource.username=root

spring.datasource.password=root

spring.datasource.driver.class=com.mysql.jdbc.Driver

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect

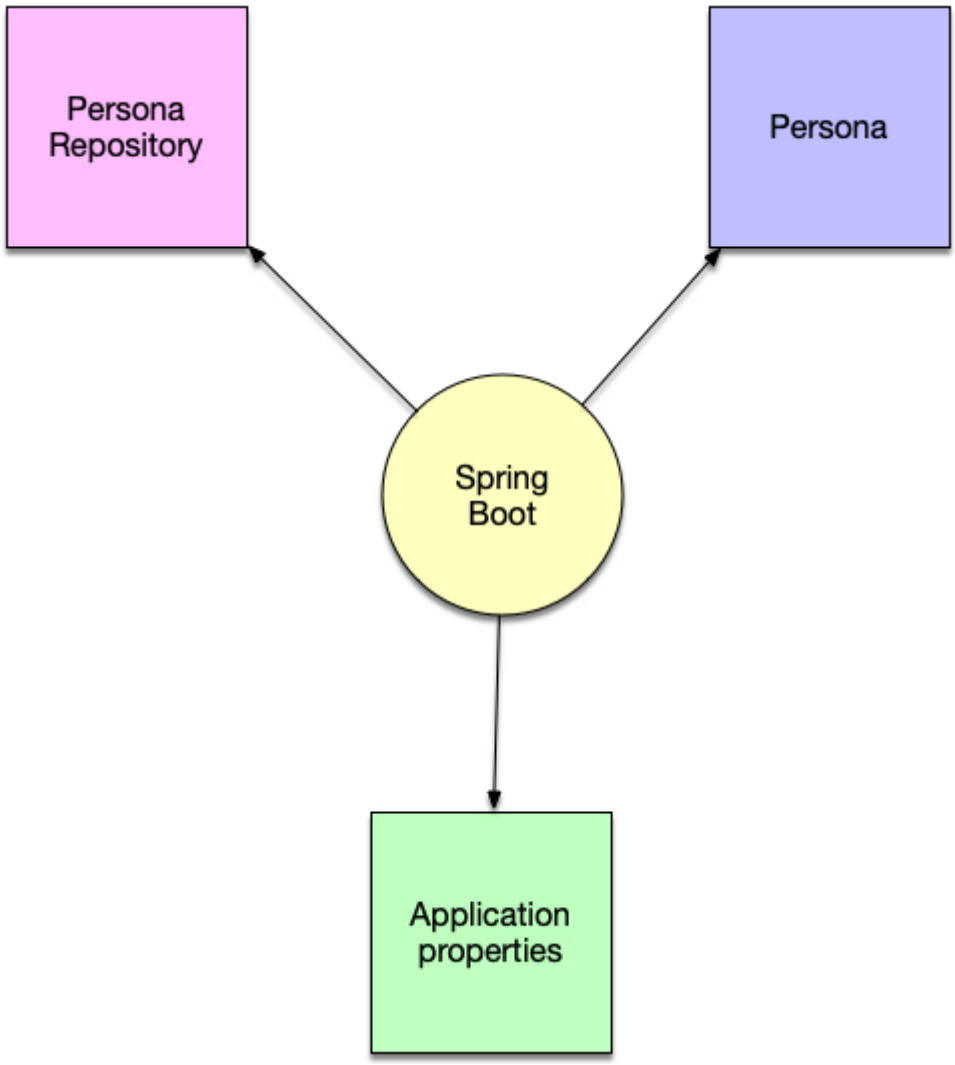
Spring Boot JPA y Repositorios

Realizado este paso recordemos que Spring nos permite de forma transparente crear interfaces de repositorios apoyados en Spring Data que nos automatiza las operaciones básicas.

```
package com.arquitecturajava.jpa.repositories;

import org.springframework.data.repository.CrudRepository;
```

```
import com.arquitecturajava.jpa.models.Persona;

public interface PersonaRepository extends CrudRepository<Persona,
String>{

}
```

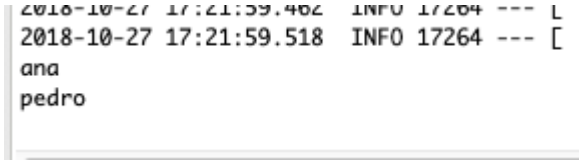
Ya disponemos de las piezas necesarias para acceder a la base de datos via Spring boot (Entidad ,FicheroConfiguracion,Repositorio)



Realizados estos pasos lo último que queda es ver cuál es el contenido del programa principal que invoca a este repositorio y nos selecciona los datos de la base de datos.

```
package com.arquitecturajava.jpa;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

import com.arquitecturajava.jpa.repositories.PersonaRepository;

@SpringBootApplication
public class JpaApplication implements CommandLineRunner {

	@Autowired
	PersonaRepository repositorio;
	public static void main(String[] args) {
		SpringApplication.run(JpaApplication.class, args);
	}

	@Override
	public void run(String... args) throws Exception {
repositorio.findAll().forEach((p)->System.out.println(p.getNombre()));
	}
}
```

Simplemente inyectamos el repositorio y ejecutamos Spring Boot desde linea de consola:



Los registros de la base de datos se imprimen en la consola , acabamos de ver como se configura Spring Boot JPA y como hacer uso de Spring Data para acceder a los datos de forma rápida:

1. ¿Qué es Spring Boot?
2. Uso de Spring Properties y encriptación
3. Spring GetMapping ,PostMapping etc
4. Spring Initializer