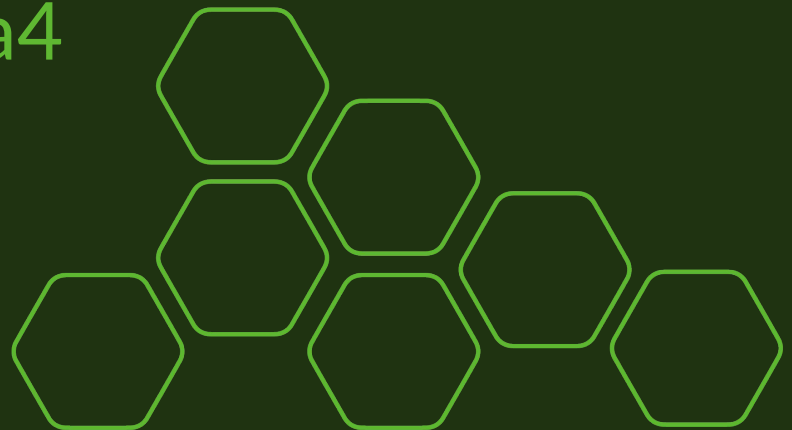


Curso de

Fundamentos de Java Spring Boot

Michael García Abelló
@maikolgarcia4



Michael García

- Software en Rappi.
- Amante de la física y las ciencias.
- Javalover.
- Desde 2013 en tech.





¿Qué deberías saber antes?

- Java 8 o superior.
- Programación orientada a objetos.
- Java funcional.



¿Qué aprenderás?

- Aprender a construir una aplicación en Spring Boot.
- Entender los conceptos de inyección de dependencias e inversión de control.
- Entender el funcionamiento de Spring Boot a nivel general.



¿Qué aprenderás?

- Qué son los beans o dependencias.
- El funcionamiento del patrón de inyección de dependencias.
- Cómo crear servicios REST.
- Implementar tu arquitectura bajo las anotaciones del proyecto.
- Mucho más...

¿Qué es Spring Boot?

- Un proyecto basado en Spring.
- Su objetivo principal es “correr” la aplicación.
- Integrar con librerías de terceros muy fácilmente.



¿Qué es Spring Boot?

- Configuraciones mínimas a nivel de properties.
- Tiene como características principales servidores web embebidos.



Características principales





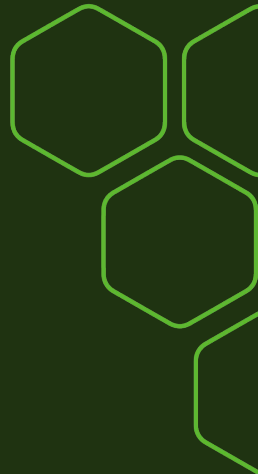
Características principales


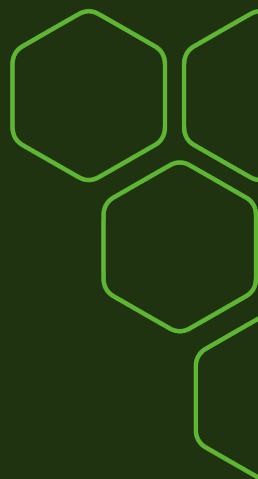
- Independiente.
- Incrustado
Jetty o Undertow.
- Proporción de dependencias.
- Sin generación de XML.
- Métricas de salud del aplicativo.

Tomcat,



Instalación de entorno de desarrollo





**¿Qué es una
dependencia?**

¿Qué es una dependencia?

- Son objetos definidos como una funcionalidad.
- Sin esta funcionalidad los otros objetos no podrán trabajar, **dependen de ellas.**



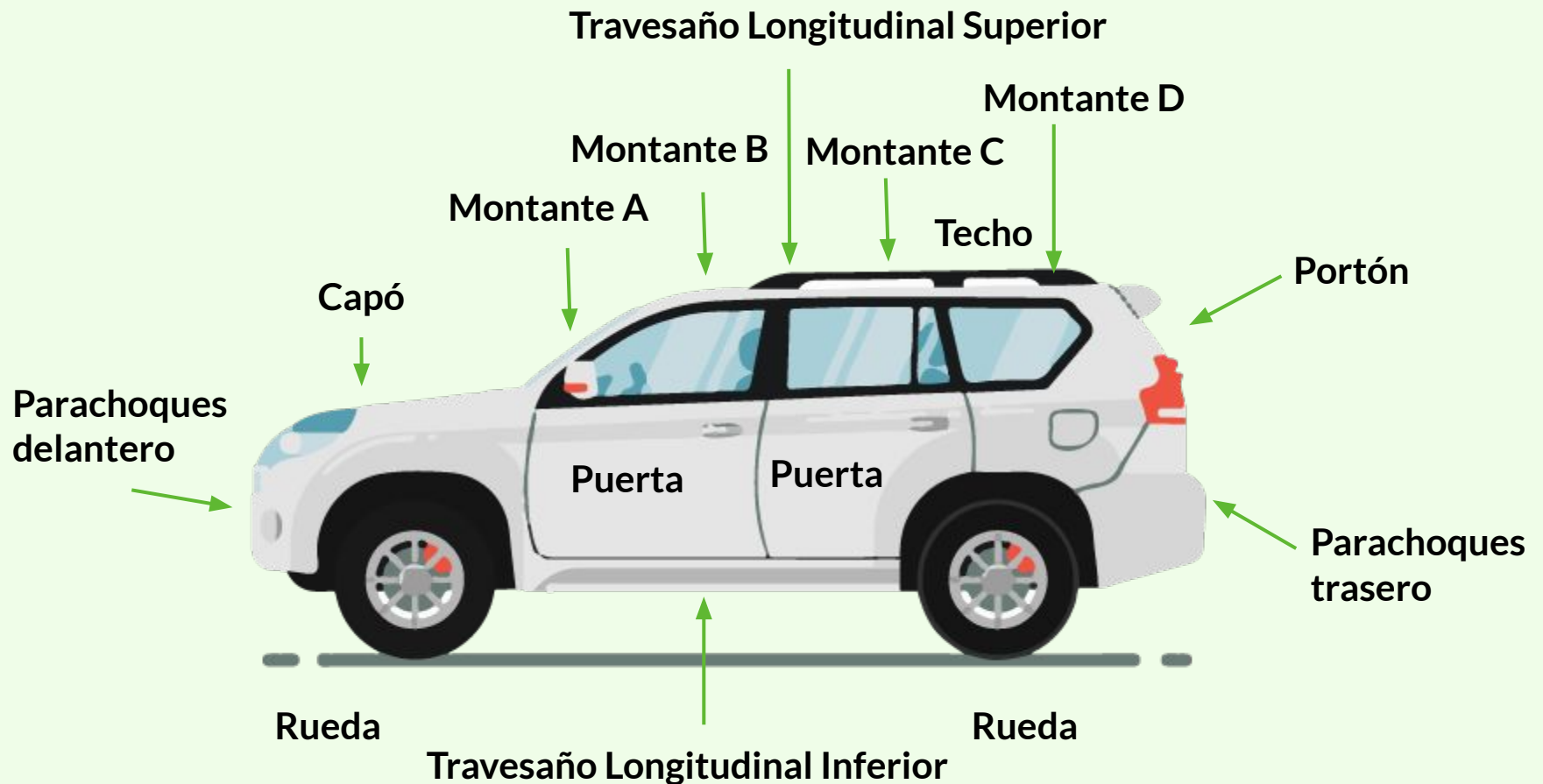
¿Qué es una dependencia?

- Necesitamos objetos que tengan otras dependencias que ayuden a cumplir un objetivo.
- Pequeña característica de un objeto específico.

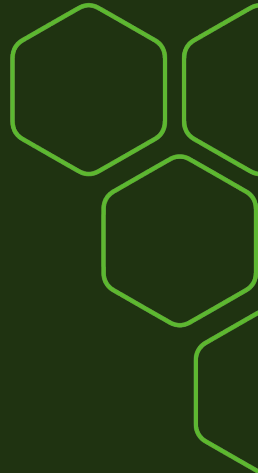




¿Qué es una dependencia?



Inversión de control y el patrón de inyección de dependencias



¿Qué es inversión de control?

- Principio que transfiere el control de objetos de un programa a un contenedor o framework.
- A diferencia con llevar el flujo de un programa de manera tradicional.

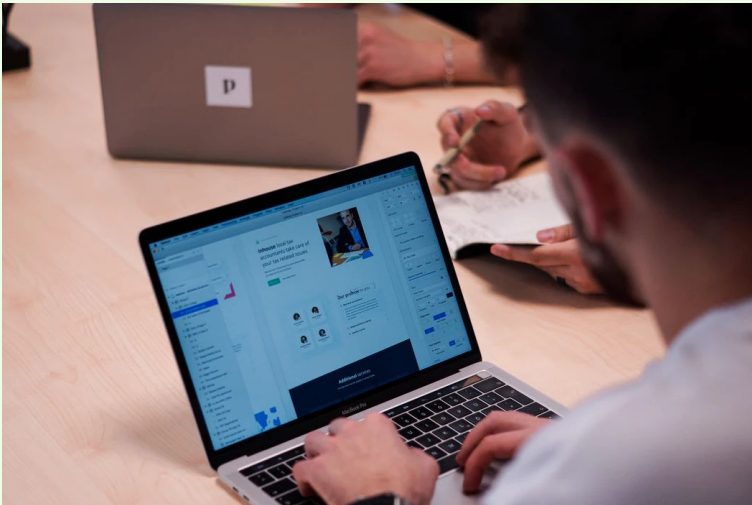


Flujo de manera tradicional



```
public class Generals {  
    public static void main(String[] args) {  
        List<Integer> lista = Arrays.asList(1, 2, 3);  
        int number = 5;  
        for(int value : lista){  
            //  
        }  
        System.out.print(number);  
    }  
}
```

🟢 ¿Qué es inversión de control?



Contenedor



Framework



Ventajas de IoC

- Desacoplamiento cuando los objetos cuentan con sus dependencias.
- Se oculta la implementación de las dependencias, beneficio de segregación de interfaces.



Ventajas de IoC

- Facilita el testing por componentes o mocks de dependencias.
- Mayor modularidad de un programa.



Ventajas de IoC

```
public interface Dependencia{
    void implementation();
}

public class Objeto implements Dependencia{
    @Override
    public void implementation(){
        //
    }
}

public class Objeto2 implements Dependencia{
    @Override
    public void implementation(){
        //
    }
}
```



IoC en el contexto de Spring Boot

- Los objetos que son administrados por el contenedor Spring IoC se denominan beans.
- Un bean es un objeto que es instanciado, ensamblado y administrado por un contenedor Spring IoC.



IoC en el contexto de Spring Boot



```
@Bean
public MyBean anyNameMethod(){
    return new MyBeanImpl();
}

}
```



¿Qué es inyección de dependencias?

- **La inyección de dependencias (DI)** es el proceso con el que los objetos definen sus dependencias.
- Código más limpio y desacoplamiento más efectivo cuando cada objeto cuenta con su dependencia.



¿Qué es inyección de dependencias?

- Implementación del principio de inversión de control.
- Definición de los otros objetos con los que trabajan.
- Clases más fáciles de probar, en particular cuando son interfaces.



DI en Spring Boot

```
    @Bean
    public MyBean myBean() {
        return new MyBeanImpl();
    }

    public class ClaseNecesitoDependencia {

        private final MyBean dependencie;

        @Autowired
        public ClaseNecesitoDependencia(MyBean dependencie) {
            this.dependencie = dependencie;
        }

        //
    }
```

Autoconfiguration y runtime





Autoconfiguration y runtime

Configura automáticamente
tus aplicaciones basadas en
dependencias JAR que tú agregaste.



Autoconfiguration y runtime

La autoconfiguración no es invasiva, siempre que queramos podemos configurar nuestros propios beans.



Autoconfiguration y runtime



Dependencia
agregada en
el proyecto.



Autoconfigura la
dependencia por ti.



Autoconfiguration y runtime

```
@Bean
public DataSource dataSource(){
    DataSourceBuilder dataSourceBuilder = DataSourceBuilder.create();
    dataSourceBuilder.driverClassName("org.h2.Driver");
    dataSourceBuilder.url("jdbc:h2:mem:test");
    dataSourceBuilder.username("SA");
    dataSourceBuilder.password("");
    return dataSourceBuilder.build();
}
```

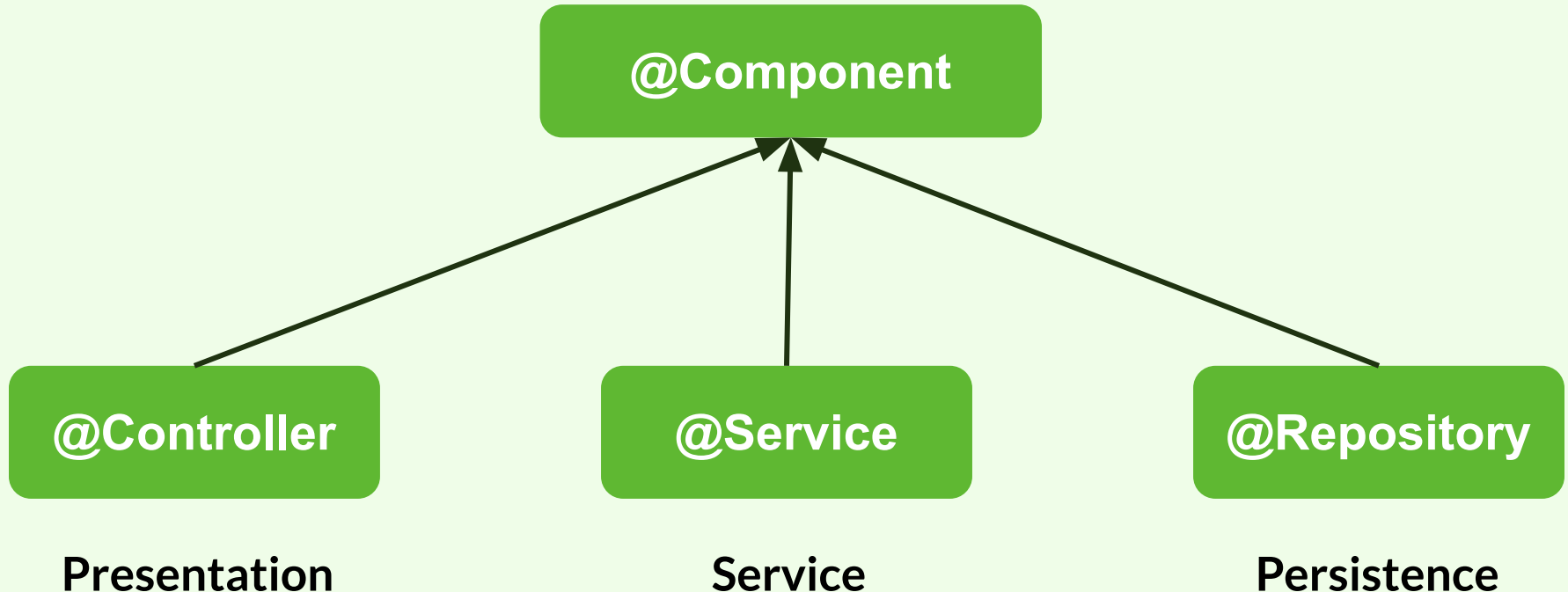
Configuración manual

Anotaciones más comunes en Spring Boot





Tipos de anotaciones





Ejemplo de inyección de dependencia

```
@Component
public class Component {

    public void myMethod(){
        //implementación general
    }
}
```

Implementación
más general

```
@Controller
public class Controller {

    public void myMethod(){
        //respuesta http
    }
}
```

Implementación
para vista o GUI



Ejemplo de inyección de dependencia

```
@Service
public class Service {

    public void myMethod(){
        //Logica de negocio
    }
}
```

Implementación
lógica de negocio

```
@Repository
public class Repository {

    public void myMethod(){
        //Persistencia de datos
    }
}
```

Implementación de
persistencia de datos



Ejemplo de inyección de dependencia

```

@Service
public class DatabaseAccountService implements AccountService {

    private final RiskAssessor riskAssessor;

    @Autowired
    public DatabaseAccountService(RiskAssessor riskAssessor) {
        this.riskAssessor = riskAssessor;
    }

    // ...
    public void miServiceFunction(){
        riskAssessor.myDependencyFunction();
    }
}
```

Creación del proyecto bajo arquitectura de dependencias



Inyección de dependencia “Component”



Ejemplo de creación de dependencia propia





Reto

- Crea tu **propia dependencia** con tus **propias implementaciones**.
- Intenta inyectar en otras capas de la aplicación.





Reto

- Intenta implementar esta dependencia en dos clases (recuerda la sobreescritura de métodos).



Cambio de puerto y path



Uso de properties y valores



Uso de properties con ejemplo de generación de POJO



¿Qué son los logs y cómo usarlos?





¿Qué son los logs?

- Error
- Info
- Debug
- Otros





Reto

- Aplica tus **propios logs** con estado **debug** o **info** para los métodos realizados dentro de las dependencias que hemos creado anteriormente.

Modelado de entidades con JPA





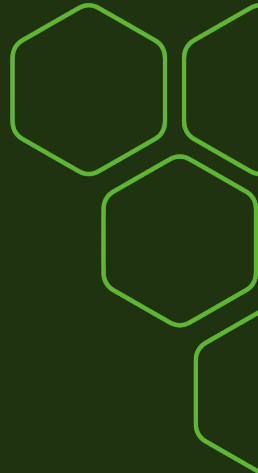
¿Qué es JPA?

Especificación de Java para **acceder, conservar y administrar** datos entre objetos o clases y una base de datos relacional.

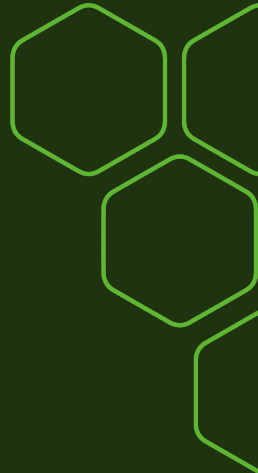
JPA

Java Persistence API

Configuración de datasource con properties y clases



Registro en base de datos con JpaRepository



Uso de JPQL en anotación Query



¿Qué es JPQL?

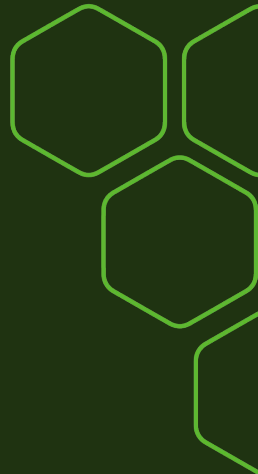
- JPQL es el lenguaje de consulta definido por JPA.
- Similar a SQL pero con la particularidad de operar sobre objetos.



Uso de anotación “valid” para apuntar a properties



Uso de Query methods





¿Qué son los Query methods?


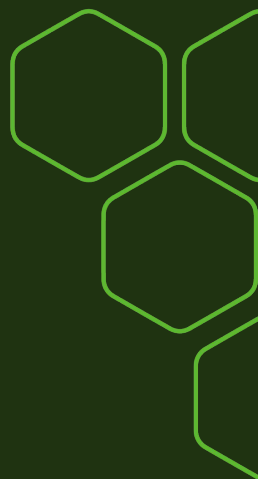
- Es una definición de una consulta manualmente como una cadena o derivarla del nombre del método.



Ejemplo de Query methods



```
public interface UserRepository extends JpaRepository<User, Long> {  
    List<User> findByLastname(String lastname);  
    User findByEmailAddress(String emailAddress);  
}
```



Uso de Query methods con Or, And, OrderBy, Between, Sort



Reto

- Crea tu **propio queryMethod** para buscar a partir de otros parámetros, ordenar de manera descendente o ascendente, utilizar las sentencias `or` `and`, `like`, etc.

Uso de JPQL con named parameters



Uso de anotación transactional



Rollback con la anotación transactional



CRUD bajo arquitectura REST



Métodos create, update y delete



Probando la API REST

Pagination con Spring Boot



Sigue aprendiendo Spring Boot





¿Qué más aprender?

- Arquitectura hexagonal.
- Comunicación entre servicios vía HTTP.
- Deploying en cloud.
- Microservicios.





¡Felicidades!

- TW: [@maikolgarcia4](#).
- GH: [github.com/maik101010](#).
- Recuerda aprobar el examen.
- Deja un review de 5 estrellas.

