

**UNIVERSIDAD PRIVADA FRANZ TAMAYO**

**HITO 3- Servicios Web**



**Nombre Completo:** Unifranz. Diego Emiliano Rivera Tapia

**Asignatura:** PROGRAMACIÓN III

**Carrera:** INGENIERÍA DE SISTEMAS

**Paralelo:** PROG (1)

**Docente:** Lic. William R. Barra Paredes

**fecha:** 27/04/2020

**github:** Servicios Web

# Índice

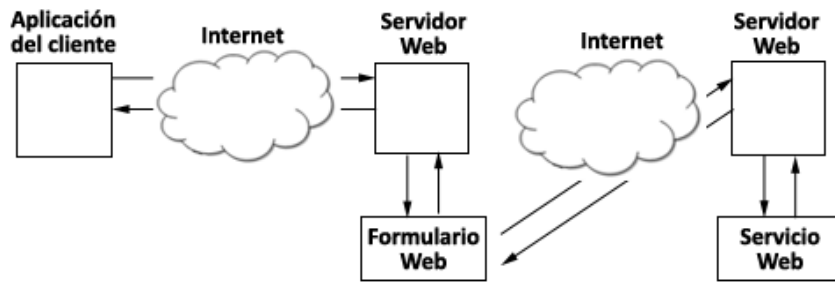
<b>ServiciosWeb.....</b>	<b>1</b>
<b>A.1 Tipos de Servicios Web.....</b>	<b>2</b>
<b>A.2 Características de SOAP.....</b>	<b>3</b>
<b>A.3 Características de REST.....</b>	<b>4</b>
<b>A.4 Comparativa entre REST y SOAP.....</b>	<b>6</b>
<b>B. Maven.....</b>	<b>7</b>
<b>B.1 Ciclo de vida.....</b>	<b>7</b>
<b>B.2 POM.....</b>	<b>8</b>
<b>B.3 El repositorio de Maven.....</b>	<b>9</b>
<b>C. Spring Framework.....</b>	<b>9</b>
<b>C.1 Módulos.....</b>	<b>10</b>
<b>C.2 Creación de contexto y beans.....</b>	<b>11</b>
<b>C.3 Spring estereotipos y anotaciones.....</b>	<b>13</b>
<b>C.4 Spring Boot.....</b>	<b>14</b>
<b>D. JPA (Java Persistence API).....</b>	<b>15</b>
<b>D.1 Anotaciones principales.....</b>	<b>15</b>
<b>E. Hibernate.....</b>	<b>16</b>
<b>E.1 Criterios.....</b>	<b>17</b>
<b>F. JWT (JSON Web Token).....</b>	<b>18</b>
<b>Web-Grafía.....</b>	<b>19</b>

## Servicios Web

Definido como un sistema software diseñado para soportar la interacción máquina-a-máquina, a través de una red y como la habilidad de dos o más sistemas o componentes para intercambiar información y utilizar la información intercambiada.

Posee una interfaz por la cual interactúa con intercambios de mensajes SOAP (Simple Object Access Protocol), generalmente usando serialización XML sobre HTTP conjuntamente con otros estándares web.

**Imagen 1:** Ejemplo del funcionamiento del servicio web.



**Elaboración:** by Roberto Jacome

**Imagen 2:** Ejemplo de servicios web.



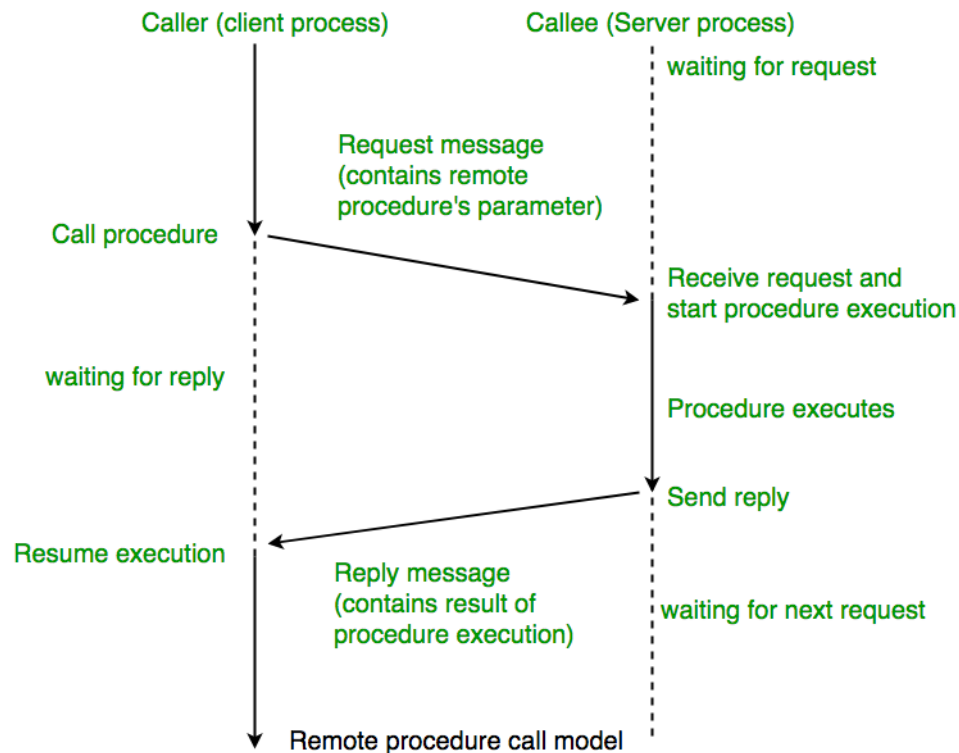
**Elaboración:** Blog WEB 2.0 Y WEB 3.0

## A.1 Tipos de Servicios Web

Los servicios web más comunes son clasificados en tres:

**Remote Procedure Calls (RPC):** Usa una interfaz que llama a procedimientos remotos y funciones distribuidas, basada en RPC. Conocida por ser una comunicación entre nodos de clientes y servidores. Funcionando de tal forma que el cliente solicita un procedimiento o función y el servidor envía una respuesta. El avance exponencial de la tecnología dejó atrás al Remote Procedure Calls, olvidándolo.

**Imagen 3:** Ejemplo del funcionamiento del RPC.

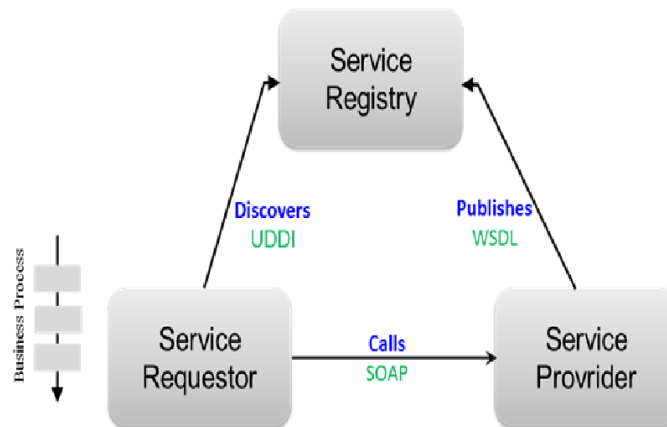


**Elaboración:** GeeksforGeeks

**Service Oriented Architecture (SOA):** Es una arquitectura de aplicación donde las funciones están definidas como servicios independientes con interfaces.

Esta arquitectura es débilmente implementada dado a que se centra en los servicios proporcionados por el documento WSDL.

**Imagen 4:** Triangulo de Service Oriented Architecture.



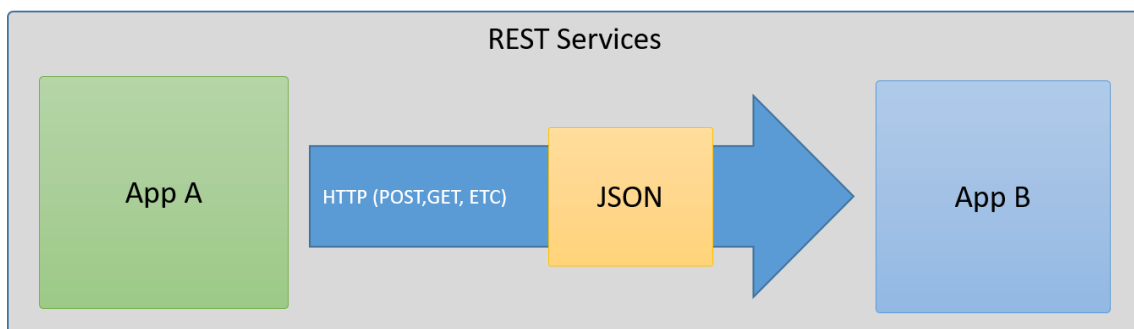
**Elaboración:** by Anders Östman

En la imagen 4 podemos observar cómo funciona la arquitectura de Service Oriented Architecture.

**REST (Representational State Transfer):** Es un nuevo enfoque de arquitectura para describir cualquier interfaz entre los sistemas que usen HTTP. El cual se usa para obtener datos o indicar la ejecución de operaciones de los datos sin importar el formato que tengan como, por ejemplo: JSON, XML, YAML, etc.

Tiene la ventaja de no usar los protocolos basados en patrones de intercambio de mensajes.

**Imagen 5:** Ejemplo del funcionamiento de REST



**Elaboración:** by Oscar Blancarte

En la imagen 5 podemos observar el funcionamiento de la arquitectura REST, como pasa información de la App A mediante verbos y un formato a la App B.

## A.2 Características de SOAP

¿Qué es SOAP (originalmente las siglas de Simple Object Access Protocol)?

SOAP es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.

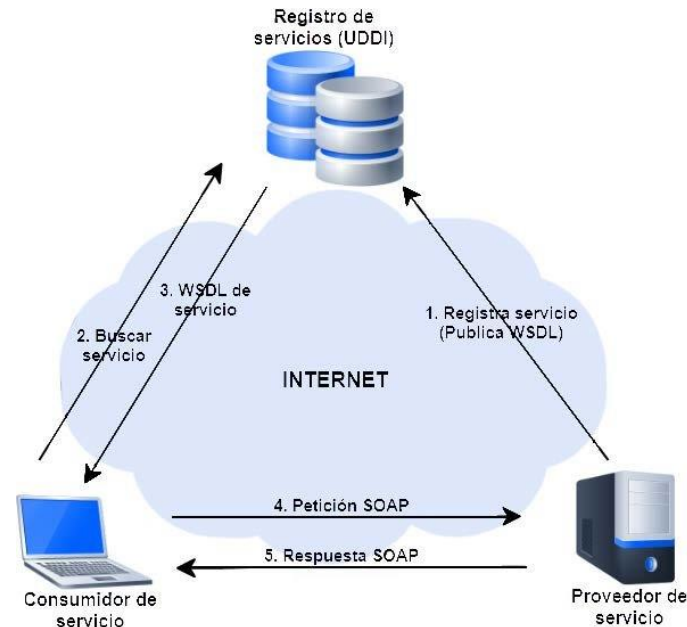
Está formado por tres partes esenciales:

- Envelope: el cual define qué hay en el mensaje y cómo procesarlo.
- Conjunto de reglas de codificación para expresar instancias de tipos de datos.
- La convención para representar llamadas a procedimientos y respuestas.

La arquitectura en los servicios web también se puede diferenciarse en tres partes:

- **Proveedor del servicio.**
- **Solicitante.**
- **Publicador**

**Imagen 6:** Funcionamiento del protocolo SOAP



Elaboración: by Alejandro Monago Ruiz

Para poder comprender la imagen 6 tenemos que saber las definiciones de WSDL y UDDI.

**WSDL (Web Services Description Language):** es el lenguaje de la interfaz pública para los servicios web. Es una descripción basada en XML de los requisitos funcionales necesarios para establecer una comunicación con los servicios web.

**UDDI (Universal Description, Discovery and Integration):** protocolo para publicar la información de los servicios web. Permite comprobar qué servicios web están disponibles.

Descripción de la imagen 6:

El proveedor de servicios envía al publicador del servicio un fichero WSDL con la definición del servicio web. El solicitante interactúa con el publicador, descubre quién es el proveedor (protocolo WSDL) y contacta con él (protocolo SOAP). El proveedor valida la petición de servicio y envía el dato estructurado en formato XML utilizando el protocolo SOAP. El fichero XML es validado de nuevo por el que pide el servicio utilizando un fichero XSD.

### A.3 Características de REST

El término fue introducido en la tesis doctoral de Roy Fielding en 2000, quien es uno de los principales autores de la especificación de HTTP.

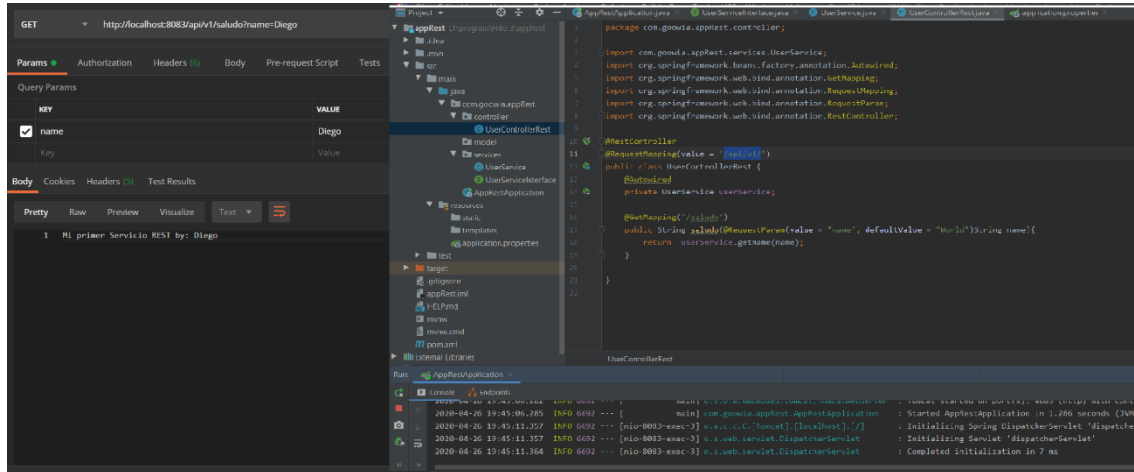
REST es el uso de formatos de mensajes extensibles, estándares y un esquema de direccionamiento global el cual explota el éxito de la web.

REST utiliza una sintaxis universal como es el uso de URIs.

Sus principales características de dividen en seis:

1. **Escalabilidad:** es un término usado en tecnología para referirse a la propiedad de aumentar la capacidad de trabajo o de tamaño de un sistema sin comprometer el funcionamiento y calidad normales del mismo.  
La variedad de sistemas y de clientes crece continuamente. Gracias al protocolo HTTP, pueden interactuar con cualquier servidor HTTP sin ninguna configuración especial.
2. **Independencia:** Los clientes y servidores pueden tener puestas en funcionamiento complejas. HTTP permite una conexión mediante las URLs.
3. **Compatibilidad:** REST como utiliza HTTP sobre Transmission Control Protocol (TCP) en el puerto de red 80 evita el ser bloqueado.  
Los servicios web se pueden utilizar sobre cualquier protocolo.
4. **Identificación de recursos:** HTTP es un protocolo centrado en URIs, donde los recursos son los objetos lógicos los cuales se comunican.
5. **Protocolo cliente/servidor sin estado:** Los mensajes al utilizar HTTP contienen toda la información para poder comprender la petición sin dificultad alguna.  
HTTP utilizan cookies y otros mecanismos para mantener el estado de la sesión.
6. **Operaciones bien definidas:** HTTP en sí define un conjunto de algunos verbos, los más importantes son POST, GET, PUT y DELETE.

Imagen 7: Ejemplo del uso de REST mediante Postman y IntelliJ Idea.



Elaboración: Propia

## A.4 Comparativa entre REST y SOAP

Para poder comprender y ver la comparación de REST y SOAP lo mejor es analizar una tabla comparativa de Alejandro Monago de su tesis “*Servicio Web API REST sobre el Framework Spring, Hibernate, JSON Web Token y BBDD Oracle*”.

REST	SOAP
<ul style="list-style-type: none"> <li>• Las operaciones se definen en los mensajes.</li> <li>• Una dirección única para cada instancia del proceso.</li> <li>• Cada objeto soporta las operaciones estándares definidas.</li> <li>• Componentes débilmente acoplados.</li> <li>• Bajo consumo de recursos.</li> <li>• Las instancias del proceso son creadas explícitamente.</li> <li>• El cliente no necesita información de enrutamiento a partir de la URI inicial.</li> <li>• Los clientes pueden tener una interfaz “listener” (escuchadora) genérica para las notificaciones.</li> <li>• Generalmente fácil de construir y adoptar.</li> </ul>	<ul style="list-style-type: none"> <li>• Las operaciones son definidas como puertos WSDL.</li> <li>• Dirección única para todas las operaciones.</li> <li>• Múltiples instancias del proceso comparten la misma operación.</li> <li>• Componentes fuertemente acoplados.</li> <li>• Fácil (generalmente) de utilizar.</li> <li>• La depuración es posible.</li> <li>• Las operaciones complejas pueden ser escondidas detrás de una fachada.</li> <li>• Envolver APIs existentes es sencillo</li> <li>• Incrementa la privacidad.</li> <li>• Herramientas de desarrollo.</li> </ul>

*Nota. Recuperado de “Servicio Web API REST sobre el Framework Spring, Hibernate, JSON Web Token y BBDD Oracle”, de Alejandro-Monago R, 2019, 25.*

	REST	SOAP
<b>Tecnología</b>	Interacción dirigida por el usuario por medio de formularios. Pocas operaciones con muchos recursos Mecanismo consistente de nombrado de recursos (URI). Se centra en la escalabilidad y rendimiento a gran escala para sistemas distribuidos hipermedia	Flujo de eventos orquestados. Muchas operaciones con pocos recursos. Falta de un mecanismo de nombrado. Se centra en el diseño de aplicaciones distribuidas.
<b>Protocolo</b>	Síncrono. XML auto descriptivo. HTTP. HTTP es un protocolo de aplicación.	Síncrono y Asíncrono. Tipado fuerte, XML Schema. Independiente del transporte. HTTP es un protocolo de transporte.
<b>Descripción del servicio</b>	Confía en documentos orientados al usuario que define las direcciones de petición y las respuestas. Interactuar con el servicio supone horas de testado y depuración de URIs No es necesario el tipado fuerte, si ambos lados están de acuerdo con el contenido. WADL propuesto en noviembre de 2006.	WSDL.  Se pueden construir automáticamente stubs (clientes) por medio del WSDL.  Tipado fuerte.  WSDL 2.0.



<b>Gestión del estado</b>	El servidor no tiene estado (stateless). Los recursos contienen datos y enlaces representando transiciones a estados válidos. Los clientes mantienen el estado siguiendo los enlaces. Técnicas para añadir sesiones: Cookies	El servidor puede mantener el estado de la conversación. Los mensajes solo contienen datos. Los clientes mantienen el estado suponiendo el estado del servicio Técnicas para añadir sesiones: Cabecera de sesión (no estándar)
<b>Seguridad</b>	HTTPS. Comunicación punto a punto segura	WS-Security Comunicación origen a destino segura.
<b>Metodología de diseño</b>	Identificar recursos a ser expuestos como servicios. Definir URLs para direccionarlos. Distinguir los recursos de solo lectura (GET) de los modificables (POST,PUT,DELETE). Implementar e implantar el servidor Web	Listar las operaciones del servicio en el documento WSDL. Definir un modelo de datos para el contenido de los mensajes. Elegir un protocolo de transporte apropiado y definir las correspondientes políticas QoS, de seguridad y transaccional. Implementar e implantar el contenedor del servicio Web.

*Nota. Recuperado de “Servicio Web API REST sobre el Framework Spring, Hibernate, JSON Web Token y BBDD Oracle”, de Alejandro-Monago R, 2019, 25 26.*

## B. Maven

Java es un lenguaje de programación orientado a objetos que se incorporó al ámbito de la informática en los años noventa.

Java contiene grandes variedades de herramientas y maven es una de ellas para la gestión y construcción de proyectos java basado en el concepto POM (Project Object Model).

Con Maven se pueden generar arquetipos, gestionar librerías, compilar, empaquetar, generar documentación.

Como se basa en patrones y estándares y trabaja con arquetipos, lo que le da a maven una libertad de crear proyectos.

### B.1 Ciclo de vida

Maven tiene un ciclo de vida dividido en tres etapas diferentes en el ciclo. El ciclo de vida por defecto es: Validación, Compilación, Test, Empaquetar, Pruebas de integración, Instalar y Desplegar.

Existen diferentes ciclos de vida como:

**Ciclo de limpieza:** clean, el cual elimina todos los ficheros.

**Ciclos de documentación:** site, genera la página web. Site-deploy, despliega la página de documentación en el servidor indicado.

## B.2 POM

Un modelo de objetos para un proyecto de maven.

“El fichero “pom.xml” es el núcleo de configuración de un proyecto Maven. Simplemente es un fichero de configuración, que contiene la mayoría de la información necesaria para construir (build) un proyecto al gusto del desarrollador”

POM está conformado por múltiples etiquetas, entre ellas <project> el cual define la arquitectura del xml.

Otros ejemplos de ficheros son:

<groupId>: suele ser el nombre.

<artifactId>: El nombre del artefacto.

<name>: nombre del proyecto.

<versión>: Versión del proyecto, suele ser 1.0.0.

<packaging>: Con este fichero se indica cómo se desea que sea empaquetado el proyecto cuando Maven lo construya.

<scope>: Que sirve para indicar a Maven cuando se quiere que utilice una biblioteca u otra.

<exclusions>: Se indica a Maven que librería heredada de las que se han incorporado, no se quiere incorporar al proyecto.

Ejemplo de un cómo crear un proyecto POM:

```
<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>clases</groupId>
  <artifactId>ProyectoMaven</artifactId>
  <version>1.0</version>
  <packaging>jar</packaging>

  <name>ProyectoMaven</name>
  <url>http://www.myjavazone.com</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>

    <dependency>
      <groupId>com.google.code.maven-play-plugin.org.playframework</groupId>
      <artifactId>jj-simplecaptcha</artifactId>
      <version>1.1</version>
    </dependency>

    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
    </dependency>
  </dependencies>
</project>
```

```

    <scope>test</scope>
  </dependency>
</dependencies>

<build>
  <finalName>WebProject</finalName>
</build>

</project>

```

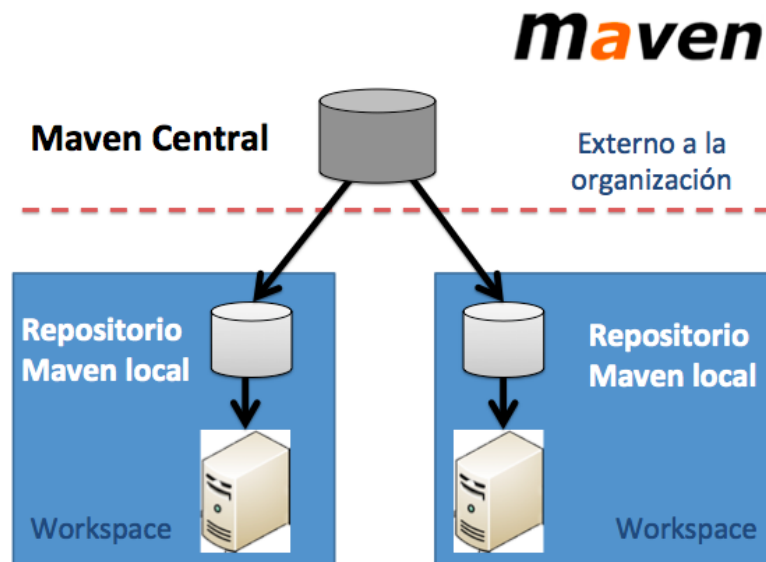
Elaboración: by MYJAVAZONE

### B.3 El repositorio de Maven

Todas las dependencias terminan guardadas en un mismo repositorio, usualmente `<USER_HOME>/m2/repository`.

El repositorio es compartido por todos los proyectos.

**Imagen 8:** Ejemplo de un repositorio de maven.



Elaboración: by JavierGarzas

### C. Spring Framework

Para entender que es spring primero debemos comprender que es un Framework, un Framework es un conjunto de clases y soluciones ya implementadas para su uso con el fin de estandarizar, agilizar y resolver los problemas.

Spring Framework facilita el desarrollar aplicaciones de manera más eficaz y rápida.

A diferencia de las aplicaciones web habituales Spring solo necesita tener configurado un servlet principal, el cual se encarga de recibir todas las peticiones de HTTP que llegan de la aplicación.

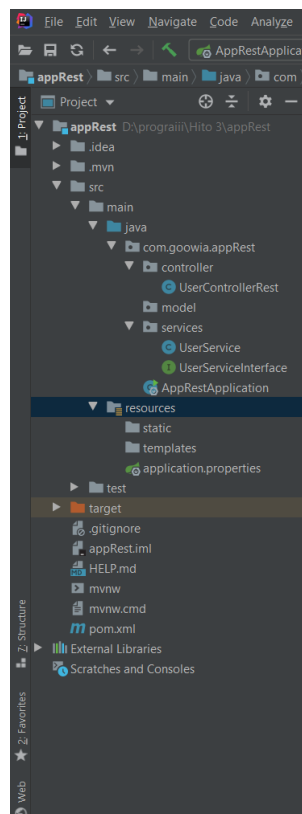
**Servlet:** Un pequeño programa residente en el servidor que generalmente se ejecuta automáticamente en respuesta a la entrada del usuario. Servlet se apoya en un contenedor que se carga en el despliegue de la aplicación.

El framework llevará a cabo la creación y destrucción de las instancias de los objetos en relación a cómo se definan en el contexto.

A continuación, se presentarán ámbitos de la creación de bean:

- **Singleton:** Es el ámbito por defecto de Spring, es decir, si no se especifica el tipo en la creación del bean, Spring lo creará con este ámbito. El contenedor de Spring creará una única instancia compartida de la clase designada por este bean, por lo que siempre que se solicite este bean se estará inyectando el mismo objeto.
- **Prototype:** El contenedor de Spring creará una nueva instancia del objeto descrito por el bean cada vez que se le solicite el mismo. En algunos casos puede ser necesario, pero hay que tener en cuenta que no se debe abusar de este tipo puesto que puede causar una pérdida de rendimiento en la aplicación.
- **Request:** El contenedor de Spring creará una nueva instancia del objeto definido por el bean cada vez que reciba un HTTP request.
- **Session:** El contenedor de Spring creará una nueva instancia del objeto definido por el bean para cada una de las sesiones HTTP y entregará esa misma instancia cada vez que reciba una petición dentro de la misma sesión.

**Imagen 9:** Ejemplo de un Spring Framework.



**Elaboración:** Propia

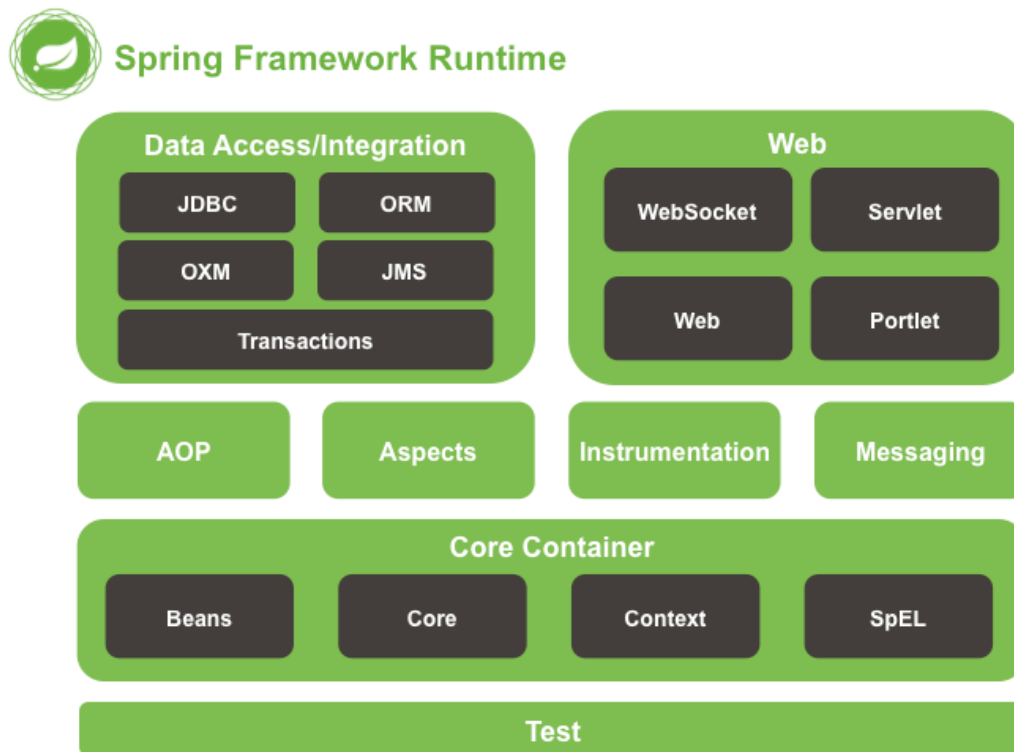
## C.1 Módulos

Cuando hablamos de módulos, hacemos referencia a una porción de programa en el ordenador. Para hacer mención a uno de los módulos más famosos, hablaremos de Core. Este módulo permite inyectar en Spring beans de cara diseño de inversión de control (IoC). El objetivo central de este

tipo de diseños es especificar respuestas y acciones deseadas ante sucesos específicos, controlando que algún tipo de arquitectura o entidad externa los ejecute.

Otro caso digno de comentar es el módulo Web, dónde lo integramos con Spring el Modelo Vista Controlador (MVC) y la definición interna de servlets. Como pudimos ver a lo largo de esta explicación, entendemos que Spring se apoya en distintos módulos según sus funcionalidades. El siguiente esquema, es una representación de los mismos:

**Imagen 10:**



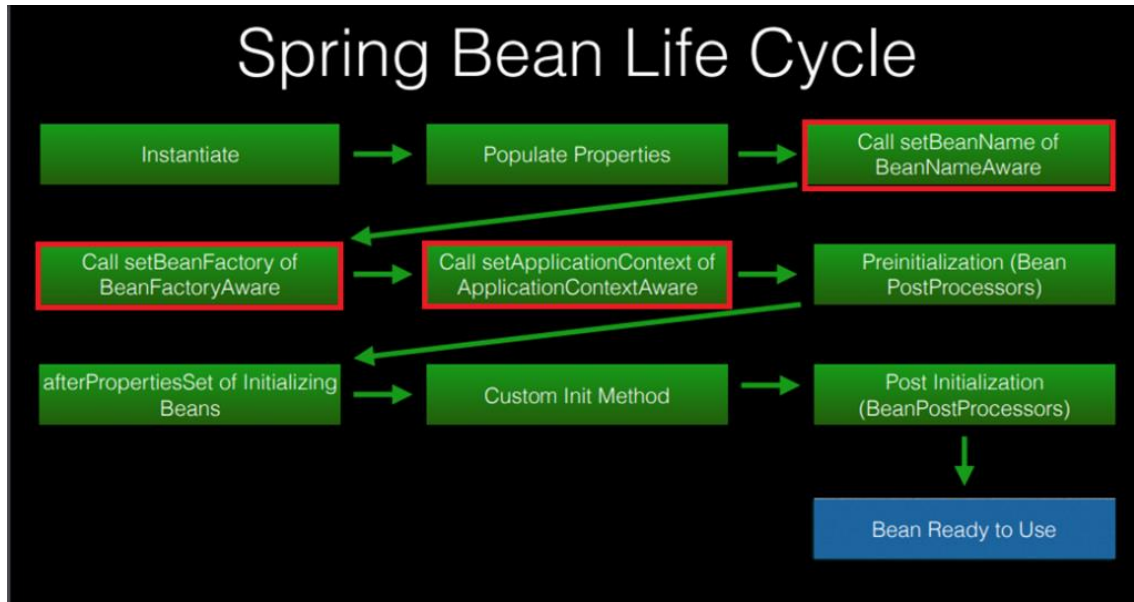
**Elaboración:** Spring

## C.2 Creación de contexto y beans

El contenedor de inversión de control (IoC) es el principal componente del framework de Spring. Dentro de sus funciones, administra los *beans* y debemos entender a un bean como un objeto de Java que depende administrativamente de Spring. Si mencionamos algunas de sus tareas más importantes, están instanciar, iniciar, destruir objetos e inyectar dependencias (DI).

Este contenedor necesita de configuración previa, y generalmente se lo hace a través del archivo XML de configuración en Spring (también podemos utilizar anotaciones o código Java). Podemos formar su cuerpo una vez que definamos el contexto. Podemos agregar beans como etiquetas, para usarlos posteriormente en cualquier parte de la aplicación. A continuación presentamos un ejemplo del ciclo de vida de un bean:

Imagen 11:



Elaboración: DZone

Un problema del fichero xml , es que si queremos declarar un gran número de componentes se puede hacer muy pesado para su manejo y su lectura. Para subsanar este tema, Spring te deja haver anotaciones en las clases para poder obtener el mismo resultado, pero trabajándolo de forma más cómoda.

A continuación, presentamos un ejemplo de cómo utilizar la etiqueta Service:

```
AppRestApplication.java x UserServiceInterface.java x UserService.java x UserControllerRest.java x ap
1 package com.goowia.appRest.services;
2
3 import org.springframework.stereotype.Service;
4
5 @Service
6 public class UserService implements UserServiceInterface{
7     @Override
8     public String getName(String name) { return "Mi primer Servicio REST by: " + name; }
9
10 }
11
12
```

Elaboración: Propia

Luego de realizar esta acción, debemos indicarle al xml que cargue todas las anotaciones creadas usando dos etiquetas especiales:

- <context:annotation-config/>: Informamos al framework que utilizaremos anotaciones de código
- <context:component-scan/>: Se indica la ruta de paquetes que debe escanear para buscar clases y utilizar los beans.

### C.3 Spring estereotipos y anotaciones

Spring tienen multitud de etiquetas y anotaciones con el fin de organizar el contenido

Según Monago los estereotipos se definen de la siguiente manera:

**@Component:** es el estereotipo principal, indica que la clase anotada es un componente o bean de Spring.

**@Repository:** Es el estereotipo que tiene como función dar de alta un bean para que implemente el patrón repositorio, que es el encargado de almacenar datos en una base de datos o repositorio de información que se necesite. Al marcar el bean con esta anotación, Spring aporta servicios transversales como conversión de tipos de excepciones.

**@Service:** Este estereotipo se encarga de gestionar las operaciones de negocio más importantes a nivel de la aplicación y aglutina llamadas a varios repositorios de forma simultánea. Su tarea fundamental es la de agregador.

**@Controller:** El último de los estereotipos, es el que realiza las tareas de controlador y gestión de la comunicación entre el usuario y el aplicativo. Para ello se apoya habitualmente en algún motor de plantillas o librería de etiquetas que facilitan la creación de páginas.

**@Autowired:** con ella es posible inyectar un componente como puede ser un servicio o un bean en la clase que se desea hacer uso de él.

Monago (2019,P.34-35) señalando y definiendo los estereotipos del Spring Framework.

**Ejemplo:** Algunas anotaciones

```
1 package com.goowia.appRest.controller;
2
3 import com.goowia.appRest.services.UserService;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.RequestMapping;
7 import org.springframework.web.bind.annotation.RequestParam;
8 import org.springframework.web.bind.annotation.RestController;
9
10 @RestController
11 @RequestMapping(value = "/api/v1/")
12 public class UserControllerRest {
13     @Autowired
14     private UserService userService;
15
16     @GetMapping("/saludo")
17     public String saludo(@RequestParam(value = "name", defaultValue = "World")String name){
18         return userService.getName(name);
19     }
20 }
21 }
```

Elaboración: Propia

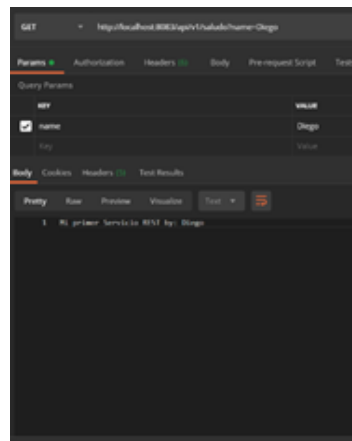
```

1 package com.goowia.appRest;
2
3 import ...
4
5
6 @SpringBootApplication
7 public class AppRestApplication {
8
9     public static void main(String[] args) { SpringApplication.run(AppRestApplication.class, args); }
10
11
12 }
13
14

```

**Elaboración:** Propia

Ejecución en Postman:



**Elaboración:** Propia

## C.4 Spring Boot

Es una infraestructura ligera que elimina la mayor parte del trabajo de configurar las aplicaciones basadas en Spring.

SpringBoot nace con la intención de simplificar los pasos 1 y 3 y que nos podamos centrar en el desarrollo de nuestra aplicación.

**Ejemplo:** Una aplicación de Spring con estructura Maven totalmente configurada.

mvnw	✓	hoy 7:29
mvnw.cmd	✓	hoy 7:29
pom.xml	✓	hoy 7:29
src	✓	hoy 8:31
main	✓	hoy 8:31
java	✓	hoy 8:31
com	✓	hoy 8:31
arquitecturaJava	✓	hoy 8:31
HolaSpringBootApplication.java	✓	hoy 7:29
resources	✓	hoy 8:31
application.properties	✓	hoy 7:29
static	✓	hoy 7:29
templates	✓	hoy 7:29
test	✓	hoy 8:31
java	✓	hoy 7:29
com	✓	hoy 7:29

**Elaboración:** arquitecturaJava

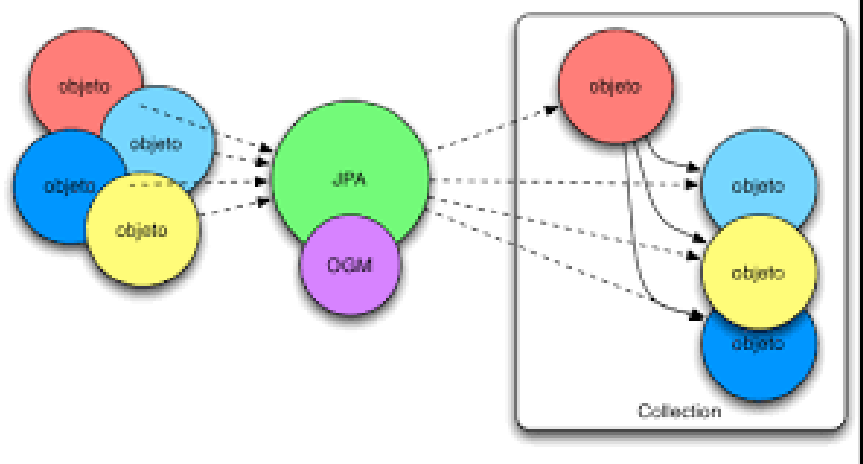


## D. JPA (Java Persistence API)

Las aplicaciones de negocios en JAVA están orientadas a objetos, pero las bases de datos relacionales vuelven la información inteligible en tablas con filas y columnas. En cuestión a las correlaciones entre ambos, necesitamos guardar la información de las bases de datos relacionales en una Interfaz que pueda funcionar dentro de la aplicación de JAVA.

JPA es una interfaz común y generalmente ésta es implementada por frameworks de persistencia. En JPA toda esta relación es transparente el desarrollador y para ello se deben crear los objetos de una manera singular, a éstos se les llama entidades. También logra conversiones de objetos y tablas esta conversión se llama ORM (Mapeo Relacional de Objetos).

**Imagen 12:** Ejemplo de JPA



**Elaboración:** Cecilio Álvarez Caules

En la imagen 11 podemos observar como el JPA interactúa entre objetos.

### D.1 Anotaciones principales

Cuando se trata de una entidad se hace referencia a un objeto POJO (Plain Old Java Object).

Las entidades JPA representan numerosas entidades. Las más importantes son:

@Entity, @Table, @Column, @Id, @JoinColumn, @OneToMany.

**EAGER:** se puede definir como un tipo de lectura temprana, es decir, en la consulta del objeto se obtienen todos los valores de las entidades que están relacionadas con la entidad. Esto es desaconsejable en el caso de asociaciones en las que se puedan ver involucrados muchos objetos, ya que puede suponer un problema de rendimiento para la aplicación.

**LAZY:** se define como lectura demorada, permite obtener un objeto de base de datos sin los valores de la relación a la que hace referencia el atributo. Es muy útil de cara al rendimiento de la aplicación ya que evita obtener ciertas propiedades que puedan no ser necesarias en la creación del objeto. Lo que se inicializaría es una asociación, de manera que, si se quiere acceder a la información en algún momento, se obtendría.

**Imagen 13:** Ejemplo de una entidad

```

@Entity
public class Customer implements Serializable {
    @Id protected Long id;
    protected String name;
    @Embedded protected Address address;
    protected PreferredStatus status;
    @Transient protected int orderCount;

    public Customer() {}
    public Long getId() {return id;}
    protected void setId(Long id) {this.id = id;}
    public String getName() {return name;}
    public void setName(String name) {this.name = name;}
    ...
}

```



Elaboración: SlidePlayer

## E. Hibernate

Hibernate es otro FrameWork el cual implementa la gestión de API, actúa de forma transparente al usuario y le aporta escalabilidad, eficiencia y facilidades a la hora de hacer consultas en la base de datos.

Hibernate tiene un doble sistema de cache el cual se divide en nivel 1 y nivel 2.

Nivel 1: Presenta automáticamente Hibernate cuando dentro de una transacción se interactúa con la base de datos. Se puede considerar como una caché de corta duración.

Nivel 2: Puede ser configurado para mejorar su rendimiento. Es válida para todas las transacciones y puede persistir en memoria durante todo el tiempo en que el aplicativo esté online.

**Imagen 14:** Ejemplo de hibernate aplicado a una sesión de fábrica.

```

34
35     public static void main(String[] args) {
36
37         //Hibernate_Empresaz.listadoDepartamentos();
38         //Hibernate_Empresaz.insertaDepartamento((byte)66, "MARKETING", "PONTEVEDRA");
39         //Hibernate_Empresaz.listadoDepartamentos();
40
41         //obtener sesion y abrir
42         SessionFactory sf = HibernateUtil.getSessionFactory();
43         Session session = sf.openSession();
44
45         /** PRUEBA METODO load() */
46         System.out.println("DATOS DEL DEPARTAMENTO 10");
47         Departamentos dep10 = new Departamentos();
48         dep10 = (Departamentos)session.load(Departamentos.class, (byte)10);
49         System.out.println("Nombre Dep: " + dep10.getDnombre());
50         System.out.println("Localidad: " + dep10.getLoc());
51         System.out.println("EMPLEADOS DEL DEPARTAMENTO 10");
52         Set<Empleados>listaEmple = dep10.getEmpleadoses();
53         Iterator<Empleados> it = listaEmple.iterator();
54         System.out.println("Numero de Empleados: " + listaEmple.size());
55         while (it.hasNext()){
56             Empleados emple = new Empleados();
57             emple = it.next();
58             System.out.println(emple.getApellido() + " * " + emple.getSalario());
59         }
60
61         /** PRUEBA METODO get() */
62         System.out.println("DATOS DEL DEPARTAMENTO 11");
63         Departamentos dep11 = (Departamentos) session.get(Departamentos.class, (byte)11);
64         if (dep11 == null){
65             System.out.println("El departamento no existe");
66         } else {
67             System.out.println("Nombre Dep: " + dep11.getDnombre());
68             System.out.println("Localidad: " + dep11.getLoc());
69         }
70     }

```

Elaboración: The Vloj

## E.1 Criteria

Es un API creado por Hibernate para facilitar las consultas a la base de datos.

Es totalmente orientada a objetos y solo hace falta crear una instancia de Criteria.

Imagen 15:

```

List cats = sess.createCriteria(Cat.class)
    .add( Restrictions.like("name", "F%")
    .addOrder( Order.asc("name") )
    .addOrder( Order.desc("age") )
    .setMaxResults(50)
    .list();

```

```

List cats = sess.createCriteria(Cat.class)
    .add( Property.forName("name").like("F%") )
    .addOrder( Property.forName("name").asc() )
    .addOrder( Property.forName("age").desc() )
    .setMaxResults(50)
    .list();

```

Elaboración: Java\_Mundoç

En la imagen 15 se puede observar un ejemplo de la aplicación de Criteria insertando nombre y año.

## F. JWT (JSON Web Token)

La autenticación basada en token es un desarrollo de aplicaciones web. El usuario se identifica con una única contraseña y aplicación web. Esta información no es almacenada en la parte del servidor esta información llega en unas cintas cifradas y cumplen con las peticiones del HTTP.

El token que envía como respuesta la aplicación tiene un tiempo de vida el cuál se debe configurar acorde a lo que interese.

El JWT está formado por tres cadenas: Header, Payload y Signature.



**Header:** Es la primera parte del token.

**Payload:** Este compuesto por los siguientes atributos:

- **iss:** especifica la tarea para la que se va a usar el token.
- **sub:** presenta información del usuario.
- **aud:** indica para que se emite el token. Es útil en caso de que la aplicación tenga varios servicios que se quieren distinguir.
- **iat:** indica la fecha en la que el token fue creado.
- **exp:** indica el tiempo de expiración del token, se calcula a partir del iat.
- **nbf:** indica el tiempo en el que el token no será válido hasta que no transcurra.
- **jti:** identificador único del token. Es utilizado en aplicaciones con diferentes proveedores.

**Signature:** La firma es la última de las tres partes, está formada por la información del Header y el Payload codificada en Base64, más una clave secreta que se configura en la propia aplicación.

**Imagen 16:**


Debugger Libraries Introduction Ask Get a T-shirt!
Crafted by  Auth0

## Debugger

ALGORITHM

HS256

**Encoded**
PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gR999IiwiaWF0IjoiYWRtaW4iOnRydWV9.TjVA950rM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
```

**Decoded**
EDIT THE PAYLOAD AND SECRET (ONLY HS256 SUPPORTED)

**HEADER: ALGORITHM & TOKEN TYPE**

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```


**PAYLOAD: DATA**

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

**VERIFY SIGNATURE**

```
hMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
)
```

☐ secret base64 encoded

 **Signature Verified**

**Elaboración:** Santi Macias

## Web-Grafía:

A continuación están los enlaces de la bibliografía necesaria para hacer este informe:

<https://idus.us.es/bitstream/handle/11441/89487/TFG-2380-MONAGO.pdf?sequence=1&isAllowed=y>

[https://www.researchgate.net/figure/Figura-36-Consumo-de-Servicios-Web-formularios-3135-Web-Services-con-JAVA-JAX-WS-es\\_fig6\\_301301481](https://www.researchgate.net/figure/Figura-36-Consumo-de-Servicios-Web-formularios-3135-Web-Services-con-JAVA-JAX-WS-es_fig6_301301481)

<http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/41>

<https://usuga9802.wordpress.com/web-2-0-y-web-3-0/>

<https://www.geeksforgeeks.org/remote-procedure-call-rpc-in-operating-system/>

[https://www.researchgate.net/figure/Service-Oriented-Architecture-triangle\\_fig1\\_268001620](https://www.researchgate.net/figure/Service-Oriented-Architecture-triangle_fig1_268001620)

<https://www.oscarblancarteblog.com/2017/03/06/soap-vs-rest-2/>

<http://www.myjavazone.com/2013/08/configuracion-del-archivo-pom-y-adicion.html>

<https://www.javiargarzas.com/2014/08/nexus-artifactory-10-min.html>