TRABAJO DE INVESTIGACION

ESTADO DEL ARTE

NOMBRE COMPLETO: Landa Garcia Daniela Claudia

MATERIA: PROGRAMACION 3

CARRERA: INGENIERIA DE SISTEMAS

DOCENTE: ING. WILLIAM RODDY BARRA

GESTION: I – 2020

3. Estado del arte:

En este informe se mostrarán las ventajas como a su vez en que consisten los móviles que es importante en el desarrollo de aplicaciones y que las hace distintas al resto.

3.1 Servicios Web:

El consorcio W3C "World Wide Web Consortium" dice que un servicio web es como un sistema software que se diseñó para soportar la interacción máquina a máquina a través de una red de forma interoperable. También tiene una interfaz la cual es posible interactuar con el mismo con el intercambio de mensajes SOAP, usando serialización XML sobre HTTP conjuntamente con otros estándares web.

3.1.1 Tipos de Servicios Web:

Se clasifican de la siguiente manera:

- Remote Procedure Calls
  "Llamadas a Procedimientos Remotos" presentan una interfaz para llamar a procedimientos remotos y funciones distribuidas. Se comunican nodo a nodo entre cliente y servidor, donde el cliente solicita que se ejecute cierto procedimiento o función y el servidor envía la respuesta.
- Service Oriented Architecture
  "Arquitectura Orientada a Servicios" es una arquitectura de aplicación en la cual todas las funciones son como servicios independientes con interfaces que pueden ser llamados para formar los procesos de negocio, al contrario que los Servicios Web basados en RPC.
- REST
  "Representacional State Transfer" es un conjunto de principios de arquitectura para describir interfaces entre sistemas que use HTTP para obtener datos o indicar la ejecución de operaciones sobre los datos, no importa el formato (XML, JSON, etc.)
  Los servicios web más usados hoy en día son: SOAP y REST.

3.1.2 Características de SOAP:

SOAP es un protocolo sobre el que se establece el intercambio de información con el servicio web. Está basado en XML y se forma por 3 partes:

- 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.

Ademas que se diferencian en 3 partes:

- Proveedor del servicio.

- Solicitante.
- Publicador.

Dentro de esta arquitectura tendría estándares importantes en esta estructuración:

- WSDL (Web Services Description Language): es el lenguaje de la interfaz pública para los servicios web. Se basa 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.

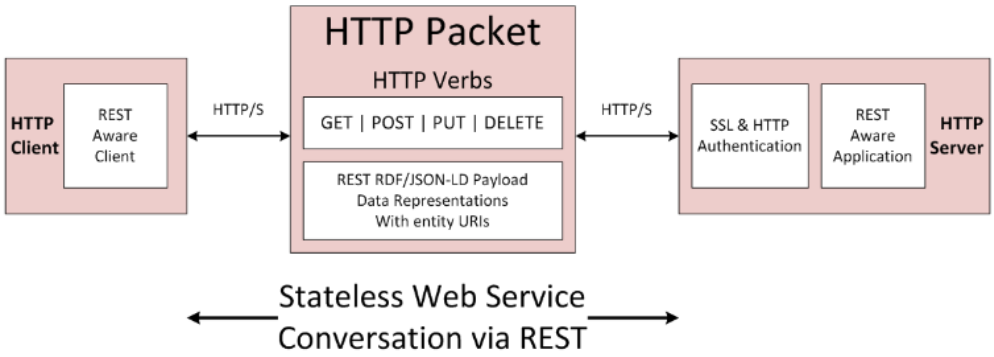
Servicio web basado en SOAP

3.1.3 Características de REST:

Las características principales de un modelo REST son:

- Escalabilidad.
- Independencia.
- Compatibilidad.
- Identificación de recursos.
- Protocolo cliente/servidor sin estado.
- Operaciones bien definidas.

Servicio web basado en REST



3.1.4 Comparativa entre REST y SOAP:

SOAP que es fuertemente acoplado, donde puede ser testado y depurado antes de poner en marcha la aplicación.

Mientras que REST su potencial se basa en su escalabilidad en todo tipo de sistemas y su escaso consumo de recursos debido al limitado número de operaciones y al esquema de direccionamiento unificado.

No.	SOAP	REST
1)	SOAP es un **protocolo** .	REST es un **estilo arquitectónico** .
2)	SOAP significa **Simple Object Access Protocol** .	REST significa **Transferencia de estado representacional** .
3)	De SOAP **no puede utilizar REST** debido a que es un protocolo.	REST **puede utilizar el SOAP** de servicios web, ya que es un concepto y se puede utilizar cualquier protocolo como HTTP, SOAP.
4)	De SOAP **utiliza interfaces de servicios para exponer la lógica de negocio** .	REST **utiliza URI para exponer la lógica de negocio** .
5)	**JAX-WS** es la API de Java para servicios web SOAP.	**JAX-RS** es la JAX-RS.
6)	De SOAP **define las normas** que deben seguirse estrictamente.	REST no define demasiado estándares como SOAP.
7)	SOAP **requiere más ancho de banda** y por que hace definicion de todo en su formato.	REST **requiere menos ancho de banda** ya que solo define su arquitectura.
8)	De SOAP **define su propia seguridad** .	Servicios web RESTful **hereda las medidas de seguridad** del transporte subyacente.
9)	De SOAP **permite XML** único formato de datos.	REST **permite diferentes** formatos de datos tales como texto sin formato, HTML, XML, JSON, etc.

3.2 Maven:

Es una herramienta para la gestión y construcción de proyectos java, basado en el concepto POM (Proyect Object Model). También se puede generar arquetipos, gestionar librerías, compilar, empaquetar, generar documentación.
Como se basa en patrones y estándares y trabaja con arquetipos, los cuales son configurables a través de su fichero protagonista, el pom.xml. Donde se puede configurar muchos aspectos de nuestro proyecto.

3.2.1 Ciclo de vida:

Maven tiene 3 ciclos de build con etapas diferenciales donde en cada uno tiene sus fases.

<u>Ciclo de vida por defecto</u>

- Validación
- Compilación (compile).
- Test (test)
- Empaquetar
- Pruebas de integración (integration-test)
- Verificar
- Instalar
- Desplegar

Ciclo de vida Maven

Ciclo de limpieza

Clean: elimina todos los ficheros generados por construcciones anteriores

Ciclo de limpieza Maven

Ciclo de documentación

Site: genera la página web de documentación del proyecto.

Site-deploy: despliega la página de documentación en el servidor indicado.

Ciclo de documentación Maven

3.2.2 POM:

POM es un modelo de objeto para un proyecto.

"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"

El pom al ser un fichero xml está compuesto por un conjunto de etiquetas.

<Project>: engloba todo el pom y en la que se define la arquitectura del xml.

```
<?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/maven-
v4_0_0.xsd">
```

<modelVersion>: Esta indica la versión del pom, para que funcione con las versiones de Maven 2 y 3 debe tener el valor “4.0.0”.

```
<modelVersion>4.0.0</modelVersion>
```

Las siguientes son las llamadas etiquetas básicas:

<groupId>: Suele ser el nombre o la web de la organización. Aunque no es necesario que tenga puntos de separación, se recomienda para actúe como paquete de Java.
<artifactId>: El nombre del artefacto.

<name>: El nombre del proyecto.

<version>: La versión de del proyecto. Por defecto se suele usar “1.0.0”. Aunque es posible usar la metodología de versiones que más nos guste.

<packaging>: Con ella se indica cómo se desea que sea empaquetado el proyecto cuando Maven lo construya. Si por ejemplo se usa como valor “jar”, se nos creará una biblioteca de Java. Otro ejemplo sería definirlo como “war”, que sería un empaquetado web para desplegar en un servidor.

```
<groupId>org.es.wsus</groupId>
<artifactId>wsus</artifactId>
<name>wsus</name>
<packaging>war</packaging>
<version>1.0.0</version>
```

<properties>: Que engloba un conjunto de otras etiquetas.

```
<properties>
<java-version>1.8</java-version>
<org.springframework-version>4.3.14.RELEASE</org.springframework-version>
<hibernate.version>4.3.6.Final</hibernate.version>
<org.slf4j-version>1.7.5</org.slf4j-version>
<jackson.databind-version>2.5.2</jackson.databind-version>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
```

<repositories>: se indica cuáles serán los repositorios en los que Maven debe buscar las librerías para nuestro proyecto.

```
<repositories>
  <repository>
    <id>central</id>
    <name>Maven Repository Switchboard</name>
    <layout>default</layout>
    <url>http://repo1.maven.org/maven2</url>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
  </repository>
</repositories>
```

<dependencies>: engloba a todas sus etiquetas hijas.

<dependency>: en las que se definen los artefactos que quiere que Maven descargue e incorpore a nuestro proyecto.

```
<dependencies>
      <dependency>

          <groupId>com.oracle</groupId>
          <artifactId>ojdbc6</artifactId>
          <version>11.2.0.3</version>
     </dependency>

<!-- Jackson -->
     <dependency>
          <groupId>com.fasterxml.jackson.core</groupId>
          <artifactId>jackson-databind</artifactId>
          <version>${jackson.databind-version}</version>
     </dependency>
<dependencies>
```

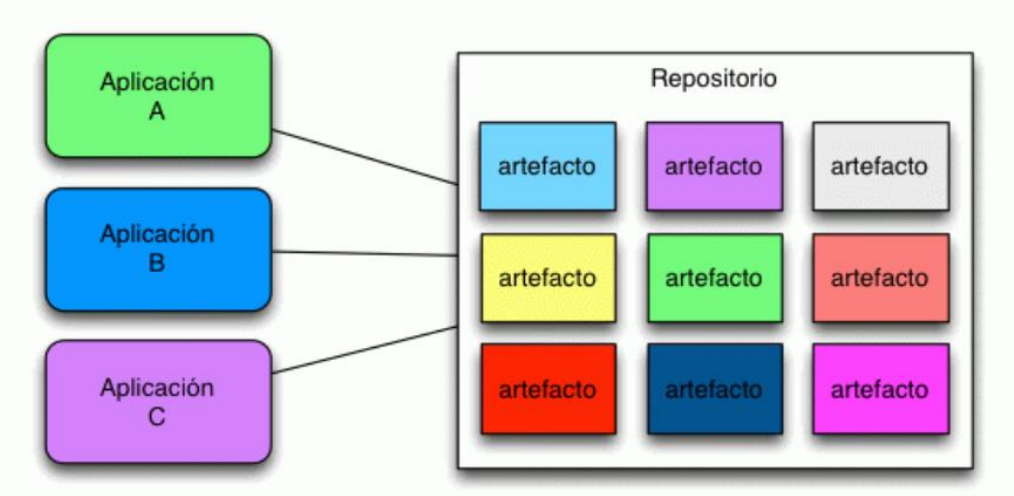
Con las etiquetas **<groupId>, <artifactId>** y **<version>** es posible identificarla biblioteca y la versión.

La etiqueta **<scope>** sirve para indicar a Maven cuando se quiere que utilice una biblioteca u otra.

Otra etiqueta muy usada es **<exclusions>**, a través de la cual se puede indicar a Maven que librería heredada de las que se han incorporado, no se quiere añadir al proyecto.

3.2.3 El repositorio de Maven

Una vez definidas todas las dependencias, Maven las descarga y las guarda todas en un mismo repositorio, por lo general <USER_HOME>/.m2/repository. Aunque se puede cambiar si se desea.

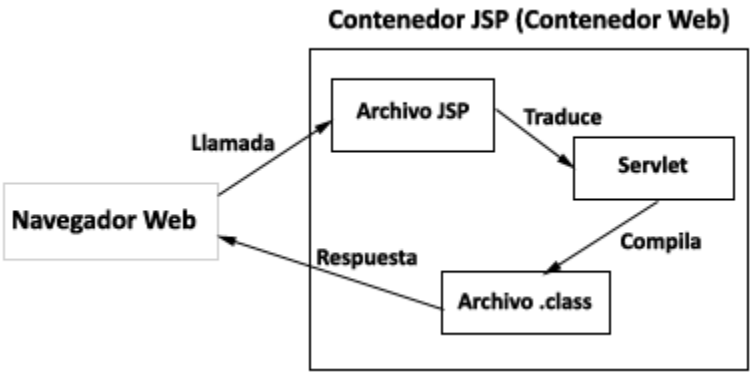


3.3

Spring Framework:

Es un modelo de aplicación web habitual sin hacer uso de algún framework en la web, el cliente hace peticiones mediante HTTP al servidor, el servidor tendrá servlets para la respuesta del cliente, estarán guardadas por un numero de servlets solicitados.

Es un conjunto de clases y soluciones para el fin de estandarizar, agilizar y resolver problemas.
El framework spring permite desarrollar aplicaciones de manera mas eficaz ahorrando líneas de código.

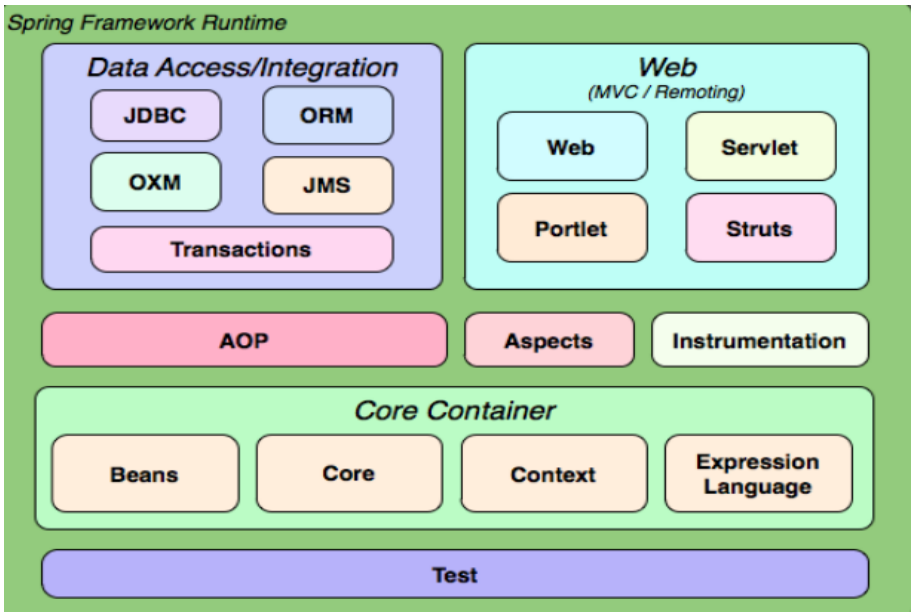


Llevará a cabo la creación y destrucción de las instancias de los objetos de relación a como se definan en el contexto. Se encontrarán los siguientes ámbitos de creación de Bean:

- **Singleton:** Es el ámbito por defecto de spring es decir que no se especifica la creación de Bean, solo spring se encargara de eso, por lo que siempre se solicita que el vean este inyectando el mismo objeto.
- **Prototye:** El contenedor de spring crea una instancia de objeto descrito por el vean cada vez que sea solicitado, en algunos casos es necesario, pero se toma en cuenta que no se debe abusar por que puede haber una perdida de rendimiento en la aplicación.
- **Request:** El contenedor de spring creara una nueva instancia del objeto definido por el vean cada ves que reciba u HTTP request
- **Session:** El contenedor de spring crea una nueva instancia del objeto definido por el vean para las sesiones HTTP y entrega la misma instancia cada vez que reciba una petición dentro de la misma sesión.

3.3.1 Módulos:

Te permite crear e inyectar beans de cara al diseño de inversión del control (IoC) que proporciona spring. Consiste en que especifica respuestas y acciones deseadas ante cualquier tipo de entidad o arquitectura externa para su ejecución, sin necesidad de preocuparse la manera en la que se realiza.
Otro modulo importante es el web, spring integra en este modulo el modelo vista



controlador (MVC) y la definición interna de servlets.

3.3.2Creacion de contextos y bean:

El principal componente del frameork spring es el contenedos de inversión de control (ioc), es el encargado para administrar los beans, un bean es un objeto Java que es administrado por spring la cual instancia, inicializa y destruye objetos, también realiza

otras tareas como la inyección de dependencias (DI). Para poder usar el contenedor se configura la forma tradicional a travez de un archivo XML de configuración de spring, es posible utilizar anotaciones y código java.
Una vez definido el contexto forma un cuerpo añadiéndole beans a través de etiquetas y pueden ser usados en otra parte de la aplicación, el XML se mostrará de la siguiente manera:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mvc"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:beans="http://www.springframework.org/schema/beans"
     xmlns:context="http://www.springframework.org/schema/context"
     xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd
          http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
          http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd" >

<beans:bean id="servicioA" class="es.prueba.ServicioA"></beans:bean>

<beans:bean id="servicioB" class="es.prueba.ServicioB"></beans:bean>

</beans:beans>
```

Se aporta una ID para acceder a ella:

```
ServicioA servicioA = (ServicioA) contexto.getBean("servicioA");
System.out.println(servicioA.mensaje());
ServicioA servicioB = (ServicioB) contexto.getBean("servicioB");
System.out.println(servicioB.mensaje());
```

Si no se declara un gran numero de variables, el fichero XML se hace pesado en el manejo y lectura. Se soluciona por el spring que da la posibilidad de usar anotaciones en clases para obtener el mismo resultado, pero más cómodo y se usa la etiqueta @Service.

```
@Service
public class ServicioA {

     public String mensaje(){
          return "Hola ServicioA";
     }

}
```

Se debe indicar al XML que cargue las anotaciones creadas, para eso se usan 2 etiquetas:

- **<context:annotation-config />** : con la que se le informa al framework que se van a usar anotaciones en el código.
- **<context:component-scan />** : se le indica al framework la ruta de paquetes que debe escanear en busca de clases para crear sus beans.

```
<context:annotation-config/>
<context:component-scan base-package="es.prueba" />
```

3.3.3 Spring estereotipos y anotaciones

Tiene muchas etiquetas y anotaciones para categorizar componentes dentro de un código.

Estereotipos:

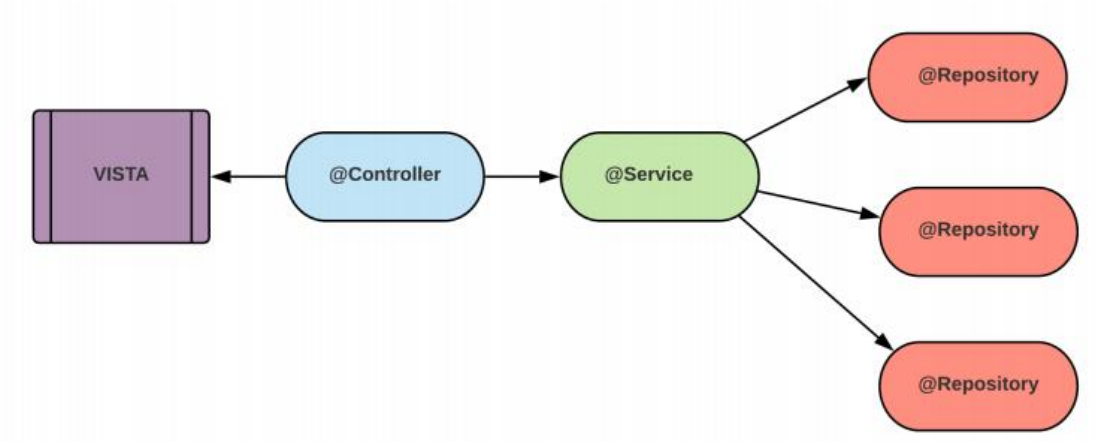
@component: Indica que la clase anotada es un componente o bean de spring

@controller @service @repository

@respository: Tiene como función dar de alta un bean para que implemente el patrón repositorio, este se encarga de almacenar datos. Al marcar el bean con esta anotación, Spring porta servicios transversales como conversión de tipos de excepciones

@service: Se encarga de gestionar las operaciones de negocio mas importantes a nivel de la aplicación y aglutina llamadas a varios repositorios de forma simultánea. Es el agregador.

@controller: Es el que realiza tareas de controlador y gestión de la comunicación entre el usuario y el aplicativo. Se apoya en algún motor de plantilla o librería de etiquetas para facilitar la creación de páginas.



Anotaciones:

@Autowired: Se puede inyectar un componente como ser un servicio o un bean.

```
public interface AlumnosSvc {

    AlumnosJson buscarAlumno(String dni) throws AppException;
    List<AlumnosJson> buscarAlumnos(List<String> listaDnis) throws AppException;
}
```

Se implementan métodos de interfaz:

```java
@Transactional
@Service
public class AlumnosSvcImpl implements AlumnosSvc {

private final Logger logger = LoggerFactory.getLogger(AlumnosSvcImpl.class);
     /**
      * Obtener Alumno por dni de BBDD
      */
     public AlumnosJson buscarAlumno(String dni) throws AppException{
          //IMPLEMENTACIÓN
     }

     /**
      * Obtener Alumnos por dni de BBDD
      */
     public List<AlumnosJson> buscarAlumnos(List<String> listaDnis) throws
AppException{
          //IMPLEMENTACIÓN
     }

}
```

Usando @autowired

```java
@Service("restService")
public class RestServiceImpl implements RestService {


     @Autowired(required = true)
      private AlumnosSvc alumnosSvc;
```

Usando la interfaz en otra clase

```java
@Service("restService")
public class RestServiceImpl implements RestService {


     @Autowired(required = true)
      private AlumnosSvc alumnosSvc;
```

Hacer uso de propiedad y métodos:

```java
alumnos = alumnosSvc.buscarAlumnos(listaDnis);
```

Si tiene métodos declarados con @service, se puede especificar la inyeccio con @qualifier

```
public interface AlumnosSvc {

@Service("obtenerDNI")
public class ObtenerDNI implements AlumnosSvc {
}

@Service("ingresarAlumno")
public class IngresarAlumno implements AlumnosSvc {
}


}
```

Hay otro recurso que serviría usando **@Resourse**

```
@Service("restService")
public class RestServiceImpl implements RestService {

     @Resource("obtenerDNI")
      private AlumnosSvc alumnosSvc;
```

Con la etiqueta @Restcontroller facilita el escenario de un API REST. Donde se debe configurar un servlet de escucha en el fichero web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/jav
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring/root-context.xml</param-value>
  </context-param>
  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>
  <servlet>
    <servlet-name>appServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>appServlet</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

Otra manera es usando @ResponseBody

```java
@Controller
public class RestController {

      @RequestMapping(value="uri")
       @ResponseBody
       public ObjetoRespuesta metodoController {
         //…
       return objeto;
       }
}
```

3.4.1 Anotaciones principales:

Una entidad se caracteriza por ser una clase de primer nivel, proporciona un constructor e implementa la interfax java.io. Entidad hace referencia a un objeto POJO (Plain Old Java Object). Cualquier entidad JPA presenta numerosas etiquetas que la caracterizan, para esto tenemos las siguientes entidades:

 @Entity: informa al proveedor de persistencia que cada instancia de esta clase es una entidad.
 @Table: permite configurar el nombre de la tabla que se está mapeando con dicha entidad. Para ello se hace uso del atributo name.
 @Colum: acompañará a cada uno de los atributos de la entidad y permitirá configurar el nombre de la columna a la que dicho atributo hace referencia dentro del sistema relacional. Esta etiqueta posee numerosos atributos, donde se puede indicar al framework propiedades que debe tener en cuenta para la columna.
 @Id: acompañará aquel atributo que permita diferenciar las distintas entidades de

manera que cada una sea única, normalmente acompaña al que se asocie a la clave primaria de la tabla que se esté mapeando.
 @JoinColumn: permite configurar aquel atributo que contiene una clave foránea a otra tabla.
 @OneToMany: esta etiqueta irá acompañando a la anterior en el caso de que el atributo de la entidad puede referenciar a más de un atributo de la tabla a la que hace referencia.
A la hora de obtener estos valores en una consulta, se pueden diferenciar dos tipos:
Eager: Se define como lectura temprana, en la consulta del objeto se obtiene todos los valores de las entidades que están en relación con la entidad.
Lazy: Se define como lectura demorada, permite que un objeto de base de datos sin los valores de la relación hacer referencia a un tributo.

3.5 Hibernate

Es un framework de persistencia de software que implementa la gestión de la API de Java. Aporta al usuario escalabilidad, eficiencia y facilidad para hacer consulta a la base de datos.
Esta diseñado para adaptarse a cualquier esquema y modelo de datos, así obtiene un modelo de tablas relacional a partir de la definición de entidades en JAVA.
Presenta un sistema de doble cache que es totalmente configurable. Se definen en nivel 1 y nivel 2.
El primer nivel: Presenta automáticamente hibernate cuando en una transacción se interactúa la base de datos. Se considera un cache de corta duración ya que es válido entre el begin y el commit de una transacción.

El segundo nivel: Puede ser configurado de cara a la mejora del rendimiento. La diferencia es que este tipo de cache es valida para todas las transacciones y persiste en la memoria durante todo el tiempo en el que el aplicativo este online y es global.

Hibernate Genera consultas SQL y libera el desarrollador mas primario, posee un lenguaje denominado HQL (hibernate query lenguaje) y de una API para que construya consultas. Denominado “criteria”

3.5.1 Criteria

Es una API creada por hibernate para facilitar consultas en una base de datos. Una de las ventajas principales es que la forma de construir querys es orientado a objetos. Es muy útil ya que da soporte a un formulario de búsqueda y nos evitara líneas de comprobación para cada campo haciendo el código mucho mas corto.

Se crea una instancia de criteria para un objeto haciendo uso de métodos de esa instancia y se da forma a las restricciones de búsqueda en torno a ese objeto.

Crit = sess.createCriteria(Cat.class);

Se limita el número de resultados:

Crit.setMaxResults(50);

Se añade criterios individuels:

Crit.add(Restrictions.like(“name”,”Fritz%”))
Crit.add(Restrictions.btween(“weight”, minWeight, maxWeight))

Para disyunciones:

crit.add(Restrictions.or(Restrictions.eq("age", new Integer(0)), Restrictions.isNull("age")))

ordenar:

crit.addOrder(Order.asc("name"))

Con todo esto se consigue cualquier consulta básica, es extenso y en esto se puede introducir código sql, crear proyecciones o formar subconsultas:

```
public List<Titulaciones> obtenerTitulacionesPorCentro(List<String> centros) throws AppException{
    List<Titulaciones> hs = null;

    try{
        Criteria criteria = sessionFactory.getCurrentSession().createCriteria(Titulaciones.class);
        criteria.createAlias("centros", "c");
        Disjunction matchDisjunction = Restrictions.disjunction();
        for (String centro : centros) {
            matchDisjunction.add(Restrictions.ilike("c.centNmNombre", centro, MatchMode.ANYWHERE));
        }
        criteria.add(matchDisjunction);
        hs = (List<Titulaciones>) criteria.list();
    }catch(Exception e){
        logger.error(e.getMessage(), e);
        throw new AppException(AppException.ERROR_10, "Ha ocurrido un error al obtener las titulaciones.",e);
    }

    return hs;
}
```

3.6 JWT (Json web token)

Esta basada en token en una referencia en desarrollo de aplicaciones web presentan una ventaja que es la de guardar sesión para los datos de un usuario.
En la autenticación con token, el usuario identifica bien con un usuario/contraseña o una ID y la web le devuelve una especie de cifra firmada para que el usuario use en las cabeceras de las peticiones HTTP.
El token se envía como respuesta a la aplicación, tiene un tiempo de vida al cual debe configurarse acorde a lo que interese.
Esta formado por 3 cadenas:

- **Header:** Es la primera parte del token, esta dormado por el tipo de token y el algoritmo de codificación utilizado:

```
{
"typ": "JWT",
 "alg": "HS512"
}
```

- **Payload:** Está compuesto por atributos llamados Claims, existen:
  - 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. Los más comunes son sub, iat y exp. Para el proyecto en cuestión, en el que no se hace login por usuario:

```
{
  "sub": "",
  "exp": 1535967018,
  "iat": 1535880618
}
```

- **Signature:** La firma es la ultima de las 3 partes esta formada por el header y el payload codificada en base 64, mas una clave secreta.

```
HMACSHA512(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) secret base64 encoded
```

Las 3:

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJzdWIiOiIiLCJleHAiOjE1MzU5NjcwMTgsImlhdCI6MTUzNTg4MDYxOH0.aaavOmuvUE2AMbkzMRef7Gx6i9IEgcYnY3XNPZXhNZGhj7L3EFjJWgyTB4j4phitRY_AGbXo1N4teshYTAn8QA