**Autenticación:**

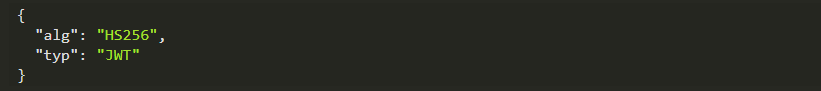
Se utiliza JSON Web Token (JWT) , crea un token que sirve para enviar enviar datos entre aplicaciones o servicios y garantiza que sean válidos y seguros.

Se utiliza cuando el usuario se quiere autenticar manda sus datos de inicio del sesión al servidor, este genera el JWT y se lo manda a la aplicación cliente, luego en cada petición el cliente envía este token que el servidor usa para verificar que el usuario esté correctamente autenticado y saber quien es.También es posible usarlo para transferir cualquier datos entre servicios de nuestra aplicación y asegurarnos de que sean siempre válido

**Estructura JWT:**

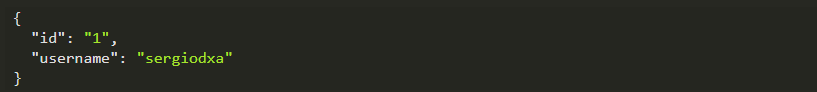
Los JWT tienen una estructura definida y estándar basada en tres partes:

1. Header: Tiene la siguiente forma:



**alg** indica el algoritmo usado para la firma y la propiedad **typ** define el tipo de token

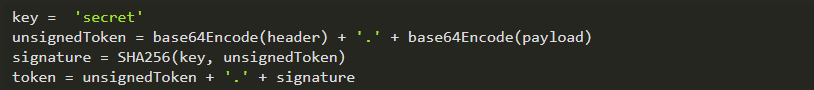
2) Payload es un JSON que puede tener cualquier propiedad ,aunque hay algunas propiedades estándar.



Propiedades estándar:

* Creador (iss) - Identifica a quien creo el JWT
* Razón (sub) - Identifica la razón del JWT
* Audiencia (aud) - Identifica quien se supone que va a recibir el JWT.
* Tiempo de expiración (exp) - Una fecha que sirva para verificar si el JWT
* No antes (nbf) - Indica desde qué momento se va a empezar a aceptar un JWT.
* Creado (iat) - Indica cuando fue creado el JWT.
* ID (jti) - Un identificador único para cada JWT.

3) Signature: se genera usando los anteriores dos campos en base64 y una key secreta para usar un algoritmo de encriptación. La forma de hacerlo entonces sería la siguiente :



**Persistencia:**

Utiliza el modela JPA , es la API de persistencia desarrollada para la plataforma Java EE. El patrón DAO tiene su lugar en una aplicación JPA y proporciona una capa delgada sobre JPA ,lo más importarte de crear un DAO por tipo de entidad que ofrece y tienen estas ventajas:

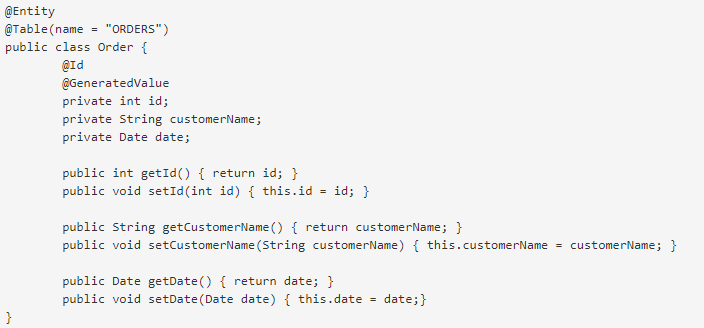
* Puede rechazar ciertas operaciones para ciertos tipos de entidades.
* En lugar de tener que elegir el método de EntityManager adecuado cada vez que quiera almacenar o cargar datos, usted decide cuál usar una vez y su equipo entero pueden cumplir con esa opción.
* Puede centralizar todas las consultas en un determinado tipo de entidad en lugar de dispersarse a través de su código.

Cuando usamos DAO:

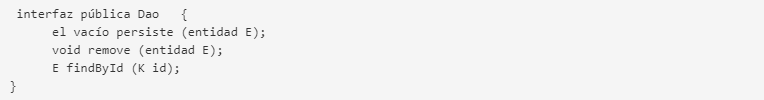
Puede escribir su JPA DAO utilizando la anotación @PersistenceContext para obtener una referencia de EntityManager .

La Clase de entidad:

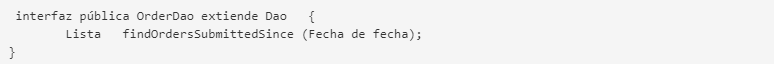
Digamos que queremos persistir en la siguiente clase de orden :



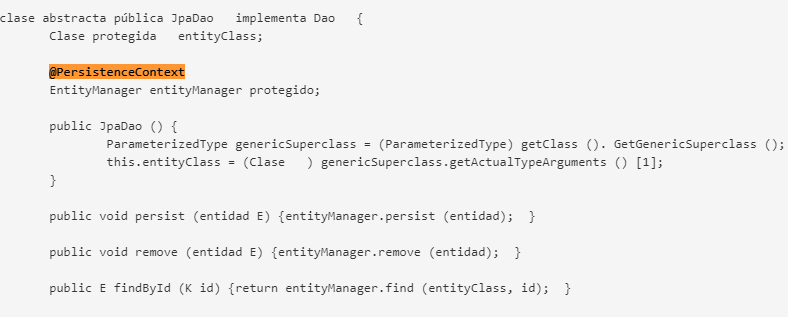
La Interfaz DAO:



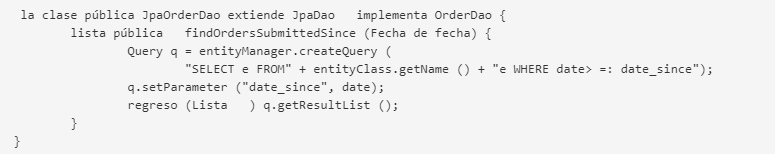
Luego definimos una subinterfaz para cada tipo de entidad que queremos que persista, agregando cualquier método específico de la entidad que deseamos.



Implementación base de DAO:



Implementación específica del DAO:



Usando el DAO:

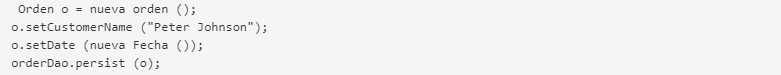
Para obtener una referencia a una instancia de su OrdenAdor usamos la siguiente notación:



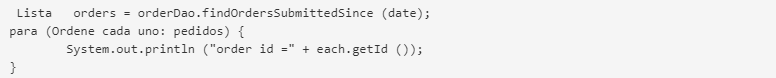
Mientras que en Spring podemos usar autowiring:



Cuando tenga la referencia al DAO lo podemos usar así:



Tambíen podemos usar la consulta específica de la entidad que agregamos a la interfaz de OrdenAdor:

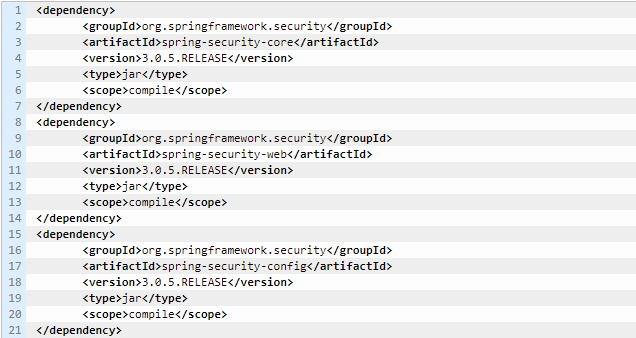


**Seguridad:**

Se utiliza Spring Security, es un framework que permitirá gestionar todo lo relativo a la seguridad de nuestra aplicación web, desde el protocolo de seguridad, hasta los roles que necesitan los usuarios para acceder a los diferentes recursos de la aplicación.

La configuración se configura mediante un fichero de configuración.

Lo primero que debemos hacer es incluir las dependencias adecuadas en nuestro pom.xml:



Ahora tenemos que habilitar el Spring Security, añadiendo un único filtro en el web.xml ,Así, éste delegará en los filtros que se creen en el fichero de configuración de Spring Security de la siguiente forma:



Después es necesario crear el archivo de configuración del Spring Security (spring-security.xml), existen dos formas de configurar Spring Security: una 100% manual y otra algo más automática.La forma manual consiste en definir todos los filtros necesarios para su correcto funcionamiento y la otra forma consiste en usar el elemento http. Este elemento configura por nosotros todos los filtros obligatorios, beans y la cadena de seguridad, si además se le pone el atributo auto-config a true, configura todos los filtros restantes necesarios para el funcionamiento.

* El primero de los beans que se crea es el springSecurityFilterChain, que es el encargado de gestionar los diferentes filtros de la cadena de seguridad.
* El filtro de autenticación (authenticationProcessingFilter), que será el que permita generar la sesión, el token, etc en el SecurityContext.
* El bean AuthenticationManager, es el encargado de recuperar la información de usuario que se está intentando loguear en el sistema.Otra propiedad que tiene es el authenticationSuccessHandler . De la forma mostrada, indicando SavedRequestAwareAuthenticationSuccessHandler, se redirigirá a la página que se solicitaba originalmente antes de la autenticación.

Ejemplo del Fichero de Configuración:



**Log:**

Utiliza Log4j es una biblioteca open source que permite a los desarrolladores de software escribir mensajes de registro, cuyo propósito es dejar constancia de una determinada transacción en tiempo de ejecución. Log4j permite filtrar los mensajes en función de su importancia. La configuración de salida y granularidad de los mensajes es realizada en tiempo de ejecución mediante el uso de archivos de configuración externos.

Niveles de prioridad de los mensajes  
Por defecto Log4J tiene 6 niveles de prioridad para los mensajes

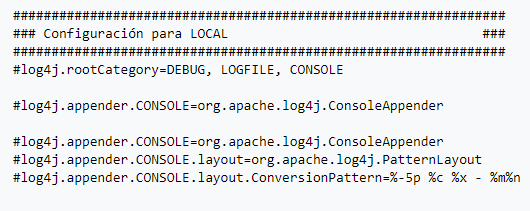
Niveles de prioridad (De menor a mayor):  
1) OFF: este es el nivel de mínimo detalle, deshabilita todos los logs.  
2) FATAL: se utiliza para mensajes críticos del sistema  
3) ERROR: se utiliza en mensajes de error de la aplicación que se desea guardar  
4) WARN: se utiliza para mensajes de alerta sobre eventos  
5) INFO: se utiliza para mensajes similares al modo "verbose" en otras aplicaciones.  
6) DEBUG: se utiliza para escribir mensajes de depuración  
7) TRACE: se utiliza para mostrar mensajes con un mayor nivel de detalle que debug.  
8) ALL: este es el nivel de máximo detalle, habilita todos los logs

Appenders:  
En Log4J los mensajes son enviados a una (o varias) salida de destino, lo que se denomina un appender.

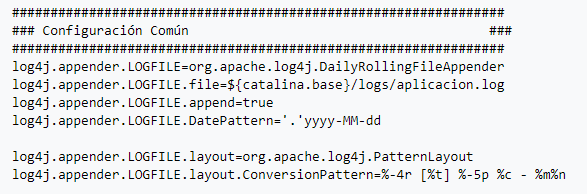
Layouts  
Permite presentar el mensaje con el formato necesario para almacenarlo simplemente en un archivo de texto .log

Configuración:

* CONSOLE imprimirá los mensajes en la consola por líneas (%m%n).

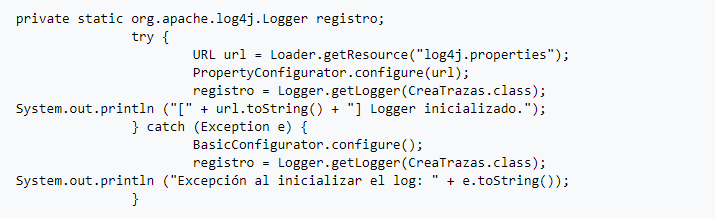


* LOGFILE añadirá (append) los mensajes a un fichero (aplicación.log), reservando los primeros 4 caracteres para los milisegundos en que se generó el mensaje (%-4r), entre corchetes quién generó el mensaje ( [%t]), cinco espacios para la prioridad del mensaje (%-5p), la categoría del mensaje (%c) y finalmente el propio mensaje junto con un retorno de carro (%m%n).



Uso:

Para utilizarlo debemos por supuesto importar las clases que necesitamos de Log4J en nuestro código.



**Manejo de Errores:**

Se utiliza Spring boot + REST:

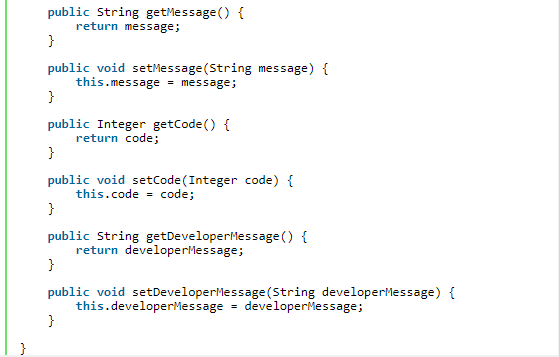
Uno de los puntos más importantes al diseñar un api rest es el manejo de errores.Implementaremos el manejo de errores utilizando Spring Boot + Jersey

1) Crear clases para representar los errores:

Crear las clases necesarias para representar los errores, en este caso se crearán dos clases ErrorMessage y ServiceException.

ErrorMessage.java





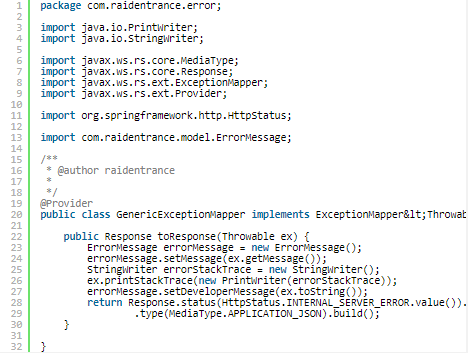
ServiceException.java



2) Crear exception mappers

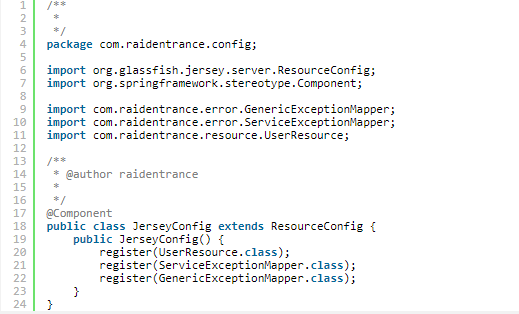
Estos definirán el que se debe hacer en caso de algún tipo de excepción.

GenericExceptionMapper.java



3) Registrando los Exception Mappers

Una vez que se crearon los Exception Mappers deben ser registrados en la aplicación. Para hacer esto se deben definir del mismo modo que los endpoints en la clase JerseyConfig.



Envío de Correo:

Spring Framework proporciona una abstracción fácil para enviar correos electrónicos mediante el uso de la interfaz JavaMailSender , y Spring Boot proporciona una configuración automática para él y un módulo de inicio.  
  
Si spring.mail.host y las bibliotecas relevantes (según lo definido por spring-boot-starter-mail) están disponibles, se crea un JavaMailSender predeterminado si no existe ninguno. El remitente se puede personalizar aún más mediante elementos de configuración del espacio de nombres spring.mail.

En particular, ciertos valores predeterminados de tiempo de espera son infinitos, y es posible que desee cambiar eso para evitar que un servidor de correo que no responda bloquee un hilo, como se muestra en el siguiente ejemplo:

