**Ejercicio Implementación de JWT mediante JJWT**

**¿Qué es JJWT?**

**JJWT** (Java JSON Web Token) es una librería/biblioteca de código abierto para **Java** y **Android** que facilita la creación y validación de tokens **JSON Web Token (JWT)**. Es gratis y de código abierto. Permite la creación de tokens con facilidad, así como la verificación de su validez utilizando algoritmos de firma como **HMAC** (Hash-based MessageAuthentication Code) o **RSA** (Rivest-Shamir-Adleman).

**JJWT** proporciona una interfaz sencilla y fácil de usar para crear y verificar **JWT**. La librería/biblioteca se encarga de la mayoría de la complejidad subyacente, como la codificación y decodificación de JSON, la firma y verificación de tokens y la gestión de claves JWT.

Existen diferentes maneras  de implementar **tokens de seguridad** como **JWT** en una aplicación desarrollada con el ecosistema de **Spring**. **JJWT** es una de ellas.

**Implementación sencilla**

**Paso 1: Crearemos un nuevo proyecto Java con Maven**

Como primera medida vamos a crear un nuevo proyecto Java con Maven desde nuestro IDE. No es necesario que sea una aplicación con Spring Boot (por el momento) dado que solo probaremos el uso de la librería JJWT.

Una vez creado el proyecto agregaremos las siguientes dependencias:

<dependency>

            <groupId>io.jsonwebtoken</groupId>

            <artifactId>jjwt-api</artifactId>

            <version>0.12.3</version> <!-- Puedes usar la versión más reciente -->

        </dependency>

        <dependency>

            <groupId>io.jsonwebtoken</groupId>

            <artifactId>jjwt-impl</artifactId>

            <version>0.12.3</version> <!-- Puedes usar la versión más reciente -->

            <scope>runtime</scope>

        </dependency>

        <dependency>

            <groupId>io.jsonwebtoken</groupId>

            <artifactId>jjwt-jackson</artifactId>

            <version>0.12.3</version> <!-- Puedes usar la versión más reciente -->

            <scope>runtime</scope>

        </dependency>

**Paso 2: Código de prueba**

Una vez agregadas las dependencias necesarias, vamos a llevar a cabo una prueba de uso de JWT en nuestra clase main. Suponiendo que nuestro proyecto se llame «**EjemploUsoApplication**«, vamos a implementar el siguiente código:

package ar.edu.centro8.ps.ejemplojjwt;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.security.Keys;

import javax.crypto.SecretKey;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

@SpringBootApplication

public class EjemplojjwtApplication {

    public static void main(String[] args) {

        //construimos extra claims

        Map<String, Object> extraClaims = new HashMap<>();

        extraClaims.put("name", "prueba jwt");

        //construimos jwt

        Date issuedAt = new Date (System.currentTimeMillis());

        Date expiration = new Date (issuedAt.getTime() + (1 \*60 \* 1000));

        String jwt;

        jwt = Jwts.builder()

                //cabecera

                .header()

                .type("JWT")

                .and()

                //payload

                .subject("pruebajwt")

                .expiration(expiration)

                .issuedAt(issuedAt)

                .claims(extraClaims)

                //firma

                .signWith(generateKey(), Jwts.SIG.HS256)

                .compact();

        //vemos nuestro jwt generado por pantalla

        System.out.println(jwt);

        SpringApplication.run(EjemplojjwtApplication.class, args);

    }

    public static SecretKey generateKey () {

        //tiene que ser larga porque dijimos que cumple con HS256

        String secretKey = "esta es mi key super segura 12345688789 HOLA que tal";

        return Keys.hmacShaKeyFor(secretKey.getBytes());

    }

}

**¿Qué hace éste código?**

* Crea un mapa (extraClaims) para añadir información adicional al token. En este caso, agrega un claim (reclamo) con clave «name» y valor «prueba jwt».
* Establece las fechas de emisión (issuedAt) y expiración (expiration) del token. En este caso, el token expirará después de 1 minuto a partir de la fecha de emisión.
* Construye el JWT usando el builder proporcionado por Jwts.builder().
* Configura la cabecera (header) del token, especificando el tipo como JWT.
* Define el contenido del token (payload) con el sujeto (subject), la fecha de expiración, la fecha de emisión y los reclamos adicionales.
* Firma el token utilizando una clave generada por el método generateKey().
  + El método generateKey() Genera una clave secreta (SecretKey) utilizando el algoritmo HMAC-SHA256 a partir de una cadena de texto definida en el método.
* Imprime el JWT generado en la consola.

En síntesis, con este código sencillo podemos cear un **JWT** con una serie de reclamos (**claims**) y firmarlo utilizando una **clave secreta generada** por un método llamado generateKey creado por nosotros mismos.

Ésta clave se utiliza para garantizar la integridad del token.

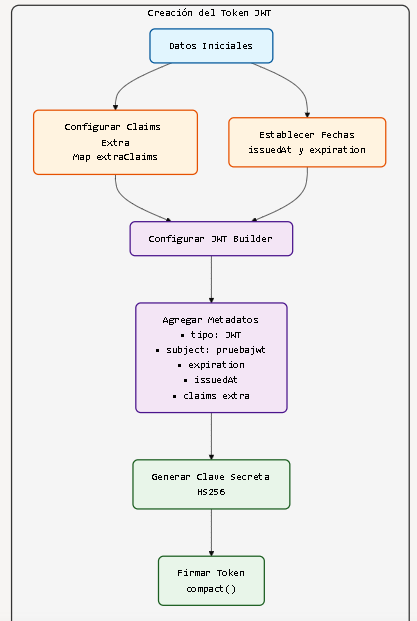
En éste ejemplo también vemos una validez corta (1 minuto) para ilustrar el concepto de expiración del token, pero en aplicaciones reales, la duración será ajustada según los requerimientos de seguridad y funcionalidad.

**Explicación Detallada**

Estructura de un JWT

Antes de profundizar en el código, es importante entender cómo está construido un token JWT. Un JWT tiene tres partes principales separadas por puntos (.)

* Header (Cabecera)
* Payload (Carga útil)
* Signature (Firma)



Como podemos ver en el diagrama, el proceso de generación del token JWT sigue una secuencia específica donde cada componente contribuye a la creación final del token. Las cajas de colores representan las diferentes fases del proceso: azul claro para los datos iniciales, naranja para la configuración, morado para la construcción del JWT y verde para la firma final.

Análisis Detallado del Código

1. Configuración Inicial

Map<String, Object> extraClaims = new HashMap<>();

extraClaims.put("name", "prueba jwt");

Esta parte del código crea un mapa donde almacenaremos información adicional que queremos incluir en nuestro token. En este caso, estamos agregando un campo "name" con valor "prueba jwt". Los claims extra son datos personalizados que podemos incluir según nuestras necesidades específicas.

2. Gestión Temporal

Date issuedAt = new Date(System.currentTimeMillis());

Date expiration = new Date(issuedAt.getTime() + (1 \*60 \* 1000));

Aquí estamos configurando dos fechas importantes para nuestro token:

issuedAt: Marca cuándo se creó el token (momento actual)

expiration: Establece cuándo expira el token (1 minuto después de su creación)

Estos timestamps son fundamentales para la seguridad del token, ya que permiten controlar su validez temporal dev.to.

3. Construcción del JWT

jwt = Jwts.builder()

.header()

.type("JWT")

.and()

.subject("pruebajwt")

.expiration(expiration)

.issuedAt(issuedAt)

.claims(extraClaims)

.signWith(generateKey(), Jwts.SIG.HS256)

.compact();

*Este es el núcleo del proceso donde construimos nuestro token JWT. Veamos cada parte:*

Configuración del Header:

.header(): Inicia la configuración del header

.type("JWT"): Especifica que este es un token JWT

.and(): Conecta con la siguiente sección

Configuración del Payload:

.subject("pruebajwt"): Establece el identificador principal del token

.expiration(expiration): Define cuando expira el token

.issuedAt(issuedAt): Registra cuando se creó el token

.claims(extraClaims): Agrega los datos adicionales que configuramos anteriormente

Firma del Token:

.signWith(generateKey(), Jwts.SIG.HS256): Firma el token usando nuestra clave secreta y el algoritmo HS256

.compact(): Convierte todo en una cadena URL-safe

4. Generación de la Clave Secreta

public static SecretKey generateKey() {

String secretKey = "esta es mi key super segura 12345688789 HOLA que tal";

return Keys.hmacShaKeyFor(secretKey.getBytes());

}

Esta función crea la clave secreta que se utilizará para firmar el token. El algoritmo HS256 requiere una clave simétrica (la misma clave se usa para firmar y verificar).

Características de Seguridad

Autenticidad: La firma asegura que nadie ha modificado el contenido del token durante su transmisión.

No Repudio: Debido a la firma digital, no se puede negar haber emitido el token.

Validación Temporal: El token incluye fechas de inicio y expiración para controlar su validez.

Estado sin Servidor: Los JWT son stateless, lo que significa que el servidor no necesita almacenar información del usuario.

Consideraciones Importantes

Seguridad de la Clave:

* La clave secreta debe ser larga y compleja
* Debe guardarse de forma segura en el servidor
* Nunca debe compartirse ni exponerse públicamente

Tiempo de Expiración:

* En este ejemplo, el token expira después de 1 minuto
* En aplicaciones reales, el tiempo dependerá del caso de uso
* Es importante encontrar un balance entre seguridad y conveniencia

Datos Sensibles:

* Aunque el token está firmado, no está cifrado
* No deben incluirse datos sensibles en los claims
* Solo se debe incluir información que sea necesaria para la autenticación