**UNIVERSIDAD PERUANA DE CIENCIAS APLICADAS**

**“AÑO DEL FORTALECIMIENTO DE LA SOBERANÍA NACIONAL”**

Logotipo

Descripción generada automáticamente con confianza media

**Ingeniería de Sistemas - 202202**

**Open Source Software - E52B**

**Profesor:**

Carlos Alberto Flores Orihuela

**Seguridad con Spring Boot y Aplicaciones Web**

**Grupo 03:**

Luigi Aguirre Charco

Wilber Alex Choque Itusaca

Jordy Peralta Ontaneda

Diana Arrieta Jeri

Shessira Cinthya Orellana Rojas

Aymin Cynthya Roman Apolinario

**LIMA – PERÚ**

**2022**

Contenido

[**Seguridad con Spring Boot y Aplicaciones Web** 3](#_Toc120637048)

[**1.** **Describir la Seguridad de Autorización y Autenticación basada en Token** 3](#_Toc120637049)

[**2.** **OAuth2.0** 4](#_Toc120637050)

[**3.** **Spring Security, JWT y CORS: Como funciona: Flujo de Autenticación** 5](#_Toc120637051)

[**4.** **Estructura JWT** 11](#_Toc120637052)

[**5.** **¿Qué es la autorización, Roles?** 11](#_Toc120637053)

[**6.** **Ejemplo de Protección de API REST con JWT FrontEnd y Backend** 13](#_Toc120637054)

[**7.** **Pruebas con Postman y Angular Frontend** 17](#_Toc120637055)

[**8.** **¿Qué es lo que más les ha impresionado o gustado?** 18](#_Toc120637056)

[**9.** **Conclusiones y Recomendaciones.** 18](#_Toc120637057)

# **Seguridad con Spring Boot y Aplicaciones Web**

Realice una investigación sobre la Seguridad en Aplicaciones Web con Spring Boot, debe contener mínimo lo siguiente:

1. **Describir la Seguridad de Autorización y Autenticación basada en Token**

En ingeniería de seguridad y seguridad informática, la autorización es una parte del sistema operativo que protege los recursos del sistema permitiendo que sólo sean usados por aquellos consumidores a los que se les ha concedido autorización para ello. Los recursos incluyen archivos y otros objetos de dato, programas, dispositivos y funcionalidades provistas por aplicaciones.

La autenticación basada en tokens es el proceso de verificar la identidad mediante la comprobación de un token. En la gestión de acceso, los servidores utilizan la autenticación por token para comprobar la identidad de un usuario, una API, un ordenador u otro servidor.

La autenticación basada en tokens puede hacer referencia a dos procesos diferentes:

* 1. **Verificar la identidad mediante un token físico.** La autenticación mediante un token físico suele tener lugar durante el proceso de inicio de sesión del usuario. El usuario tiene que demostrar que posee un objeto que nadie más tiene. Puede demostrarlo introduciendo un código mostrado por el objeto, conectando el objeto a un dispositivo mediante USB, conectando el objeto mediante Bluetooth o por varios otros métodos. Se utilizan dos tipos de tokens para este tipo de autenticación: tokens blandos tokens duros.
* **Los tokens blandos** implican introducir un código secreto o un mensaje enviado a un dispositivo para demostrar la posesión del mismo. Suele ser un código enviado a un teléfono inteligente a través de un mensaje de texto.
* **Lo tokens duros** son elementos de hardware que el usuario conecta directamente a un ordenador o dispositivo móvil para iniciar sesión.
  1. **Corroborar la identidad mediante un token web**. Los tokens web son completamente digitales. Un token web es generado por un servidor y enviado a un cliente. El token se adjunta a cada solicitud del cliente para que el servidor conozca la identidad del cliente y sepa a qué datos puede acceder.

Muchos desarrolladores son partidarios de utilizar cookies en su lugar. Los tokens web se pueden utilizar mejor para la autenticación de puntos finales de la API o para validar una conexión entre servidores, en lugar de entre el servidor y el cliente.

**Fuente:**

* Autorización. (2021, 27 de mayo). Wikipedia, La enciclopedia libre. Fecha de consulta: 02:24, noviembre 28, 2022 desde <https://es.wikipedia.org/w/index.php?title=Autorizaci%C3%B3n&oldid=135863144>.
* https://www.cloudflare.com/es-es/learning/access-management/token-based-authentication/#:~:text=%C2%BFQu%C3%A9%20Es%20La%20Autenticaci%C3%B3n%20Basada%20en%20Token?

1. **OAuth2.0**

[OAuth 2.0](https://tools.ietf.org/html/rfc6749) , que significa "Autorización abierta", es un estándar diseñado para permitir que un sitio web o una aplicación acceda a los recursos alojados por otras aplicaciones web en nombre de un usuario. Reemplazó a OAuth 1.0 en 2012 y ahora es el estándar industrial de facto para la autorización en línea. OAuth 2.0 proporciona acceso consentido y restringe las acciones de lo que la aplicación del cliente puede realizar en los recursos en nombre del usuario, sin compartir nunca las credenciales del usuario.

Aunque la web es la plataforma principal para OAuth 2, la especificación también describe cómo manejar este tipo de acceso delegado a otros tipos de clientes (aplicaciones basadas en navegador, aplicaciones web del lado del servidor, aplicaciones nativas/móviles, dispositivos conectados, etc.

OAuth es un estándar abierto que permite la autorización segura mediante el uso de un API. En la actualidad se usa en su versión OAuth 2.0, donde sus principales funciones son que proporciona flujos de autorización para aplicaciones web, de escritorio y teléfonos móviles. Servicios como Google, Facebook, Azure Active Directory, Github solo admiten el protocolo OAuth 2.0.

Su funcionamiento consiste en delegar el permiso de autenticación del usuario al servicio que gestiona dichas cuentas, de modo que es el propio servicio el que otorga acceso a las aplicaciones de terceros. Esto va a facilitar el proceso y acortar los tiempos. Se considera un estándar seguro y multiplataforma.

En el protocolo OAuth se podría definir los siguientes roles:

* **Cliente** sería la aplicación que quiere acceder a la cuenta de usuario de un determinado servicio, como Facebook, Twitter, Google, etc. Por ejemplo, si instalamos una aplicación en el móvil y nos solicita permisos para ver nuestros datos en alguna de estas redes sociales o plataformas. De esta forma evitaremos tener que rellenar todo el contenido nuevamente y lo cogerá de forma automática.
* **Usuario** es quien autoriza a la aplicación a acceder a su cuenta, mediante una ventana emergente que pide autorización, y normalmente se incluye información sobre los datos que se van a compartir al servicio nuevo. Cuando intentamos vincular cualquier programa con Facebook o Twitter, por ejemplo, tendremos que autorizar a la aplicación que pueda ceder esa información.

Es por tanto una parte importante en todo este proceso que utiliza el protocolo OAuth 2.0. Vamos a necesitar de autorizarlo para que se lleve a cabo con total garantía.

* **Servidor** de autorización recibe las peticiones de acceso de aplicaciones que desean usar al inicio de sesión de algunos de los servicios como Facebook, Twitter o Google, por ejemplo, para iniciar sesión en alguna página web, juego, etc. Este servidor se encarga de verificar la identidad del usuario y del servicio que solicita acceso, permitiendo o denegando el acceso.

Diagrama, Texto

Descripción generada automáticamente

**Fuente:**

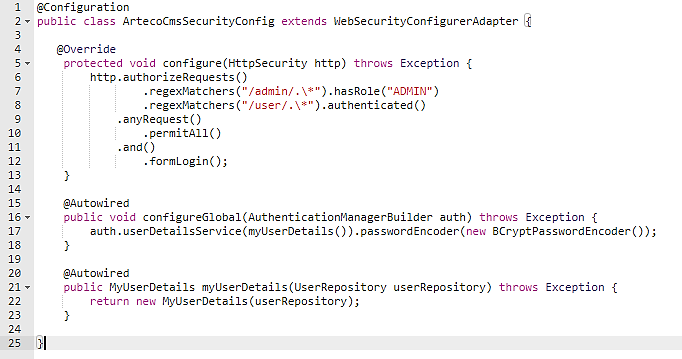
* https://auth0.com/intro-to-iam/what-is-oauth-2/
* https://www.redeszone.net/tutoriales/seguridad/que-es-oauth/

1. **Spring Security, JWT y CORS: Como funciona: Flujo de Autenticación**
   1. **Qué es Spring Security**

Spring Security es un framework de apoyo al marco de trabajo Spring, que dota al mismo de una serie servicios de seguridad aplicables para sistemas basados en la arquitectura basados en J2EE, esto permite incorporar seguridad de acceso a las aplicaciones con controles de acceso por URL entre otras muchas opciones y es más que suficiente para proteger tu programa.

El control de acceso permite limitar las opciones que pueden ejecutar un determinado conjunto de usuarios o roles sobre la aplicación. En esta dirección, Spring Security controla las invocaciones a la lógica de negocios o limita el acceso de peticiones HTTP a determinadas URLs.

Spring Security es el método más conveniente para incorporar una capa de seguridad en donde se desea que sólo algunos usuarios tengan acceso a métodos y controladores de una aplicación Spring en base al uso de roles de usuario.



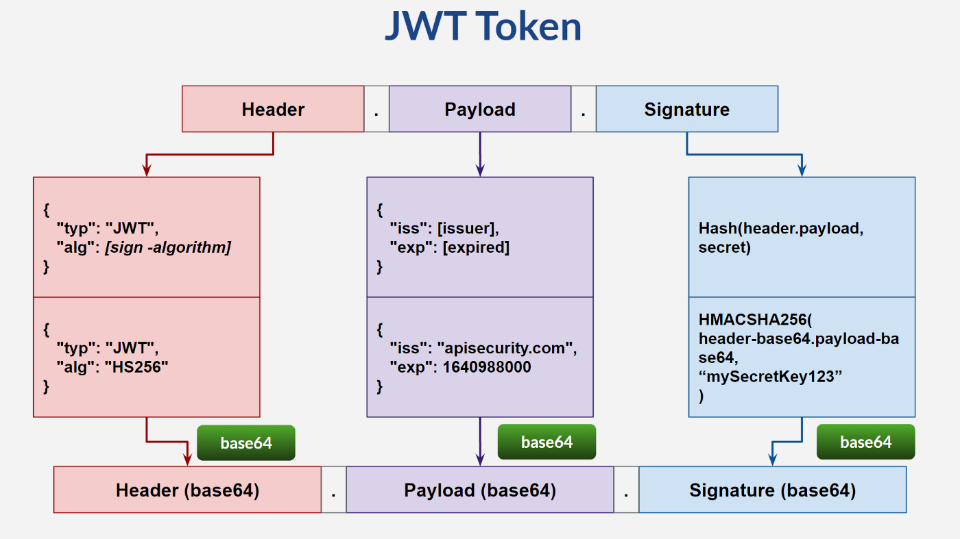
**Ilustración 1** - Spring Security basic configuration

* 1. **Que es JWT (JSON Web Token)**

JWT es un standard RFC 7519 que define un método compacto, autónomo y seguro para transmitir información entre 2 sistemas. Se usa un objeto JSON de forma segura y mucho más ligera que XML.

Característica:

* Compacto: Es de tamaño pequeño, se puede usar en un HTTP header, como parámetro GET o POST.
* Autónomo: El payload contiene toda la información necesaria para el caso de uso. En autenticación por ejemplo puede tener toda la información del usuario sin tener que consultar la base de datos.

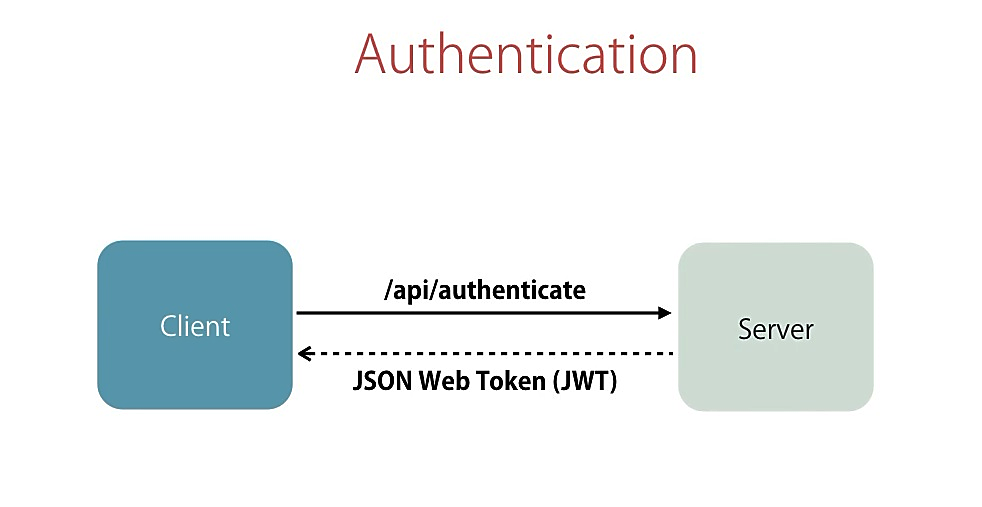


**Ilustración 2** - JSON Web Token definition

* 1. **Cómo funciona JWT**

**Autenticación**

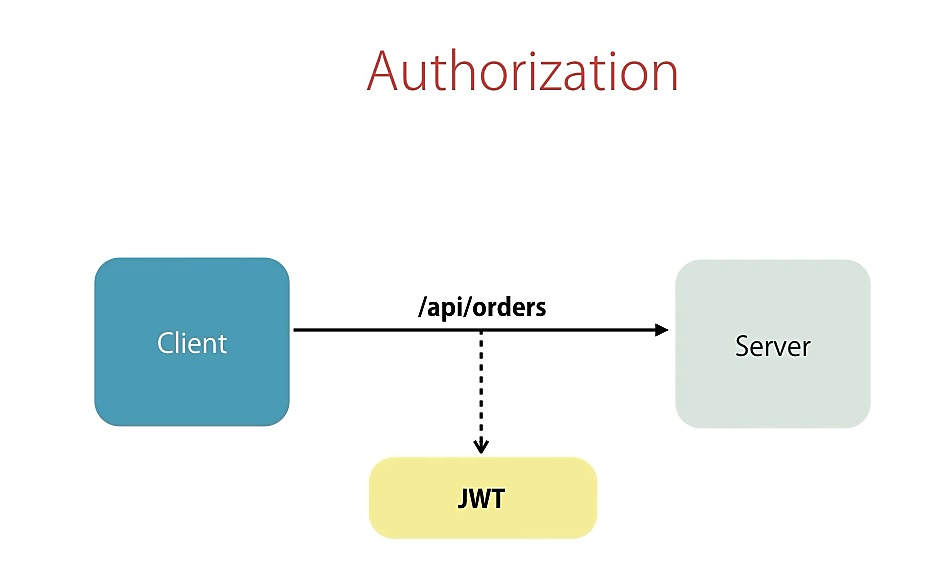
Cuando un usuario se quiere identificar en una aplicación. Se comunica con el servidor mediante un API. En el servidor se comprueban las credenciales y si son correctas se devuelve un JWT. Este JWT es almacenado de manera local en el navegador de forma que es accesible desde el cliente. Contiene información sobre el usuario que es usada para tener acceso a los servicios.



**Ilustración 3** - JWT Authenticacion

**Autorización**

Cuando un usuario entre en una zona restringida de la aplicación. El JWT se manda al servidor en Header de la petición. De esta forma el servidor puede comprobar si el usuario está autorizado para realizar la petición sin necesidad de volver a realizar una petición a la base de datos.

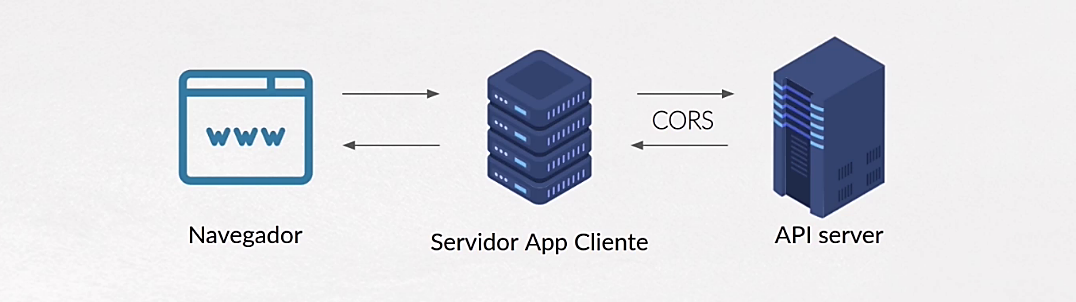
.

**Ilustración 4** - JWT authorization

* 1. **Que es CORS (Cross-Origin Resource Sharing)**

CORS es una especificación del consorcio W3C que está implementada por casi todos los navegadores, de forma que podemos especificar qué dominios están autorizados y cuáles no, y para qué están autorizados.

CORS es una política de seguridad que establece qué es ser seguro, y que nos va a dar una serie de reglas de control de acceso y de autenticación.



**Ilustración 5** - CORS relationship browser/client/api

**Ventajas del uso de CORS**

* Se puede hacer que nadie pudiera acceder a nada salvo un conjunto de orígenes, a los cuales le demos permiso por la propia URL.
* Se pueden deshabilitar todos los métodos menos GET, para que cualquiera pueda venir a hacer peticiones GET pero no pueden producir información.
* Podemos dejar GET y POST pero a deshabilitar PUT y DELETE.

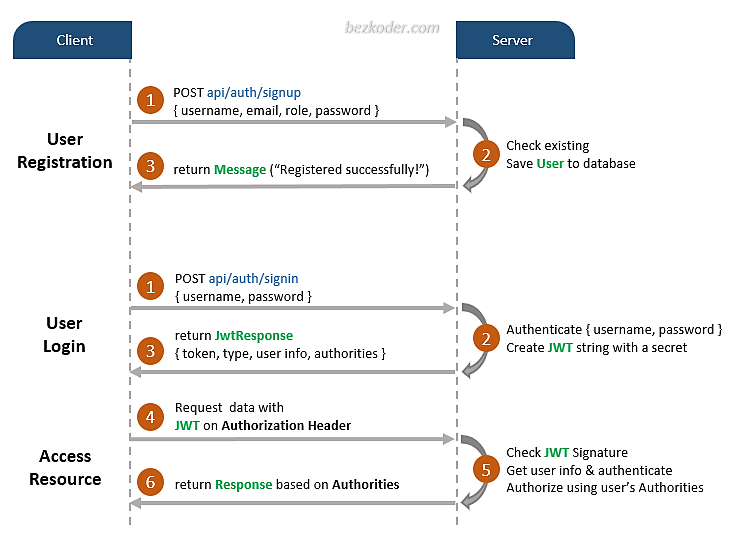


**Ilustración 6** - CORS method level configuration

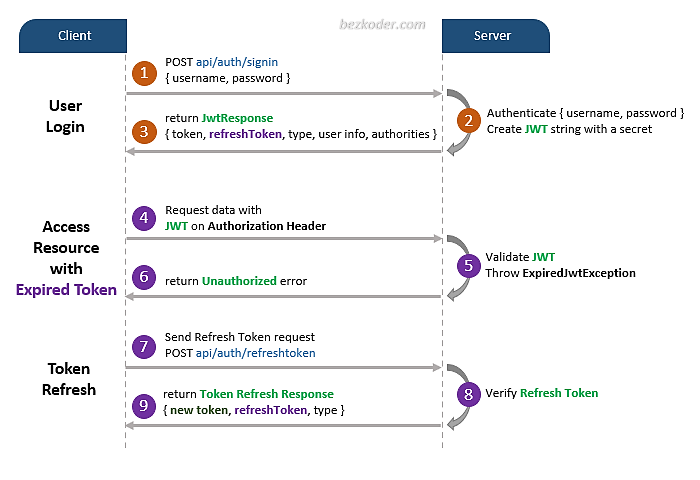


**Ilustración 7** - CORS global/class configuration

* 1. **Flujo de autenticación y autorización con Spring Security**



**Ilustración 8** - Iimplement User Registration, User Login and Authorization



**Ilustración 9** - Access protected resources

**Fuente:**

* (Cómo Configurar CORS En Una API Rest Con Spring Boot, 2019) Cómo configurar CORS en una API Rest con Spring Boot. (2019, November 26). OpenWebinars.net. https://openwebinars.net/blog/como-configurar-cors-en-mi-api-rest-con-spring-boot
* Cecilio Álvarez Caules, & Cecilio Álvarez Caules. (2018, November 29). Spring REST CORS y su configuración. Arquitectura Java. <https://www.arquitecturajava.com/spring-rest-cors-y-su-configuracion>
* Sanchez, D. (2017, December 20). JWT - JSON Web Tokens - Qué son, cómo funcionan y su estructura Yo soy Dani - desarrollo de páginas web, aplicaciones y esas cosas modernas desde Almería. Yo Soy Dani - Desarrollo de Páginas Web, Aplicaciones Y Esas Cosas Modernas Desde Almería; Yo soy Dani - desarrollo de páginas web, aplicaciones y esas cosas modernas desde Almería. https://yosoydani.com/jwt-json-web-tokens-funcionan-estructura/
* Conoce la estructura de JWT (JSON Web Token). (2021). Hostgator.mx. <https://www.hostgator.mx/blog/estructura-de-jwt/>
* Getting Spring Security: Spring Security. (2022). Spring.io. https://docs.spring.io/spring-security/reference/getting-spring-security.html

‌

1. **Estructura JWT**

Los JWT están compuesto por 3 partes:

* Cabecera (Header) : Se trata de unos datos básicos estandar.
* Carga útil (Payload): Es el cuerpo principal donde se almacenan los datos que interesan transmitir.
* Firma de verificación (Verify signature): Es una firma para verificar la autenticidad de los datos.

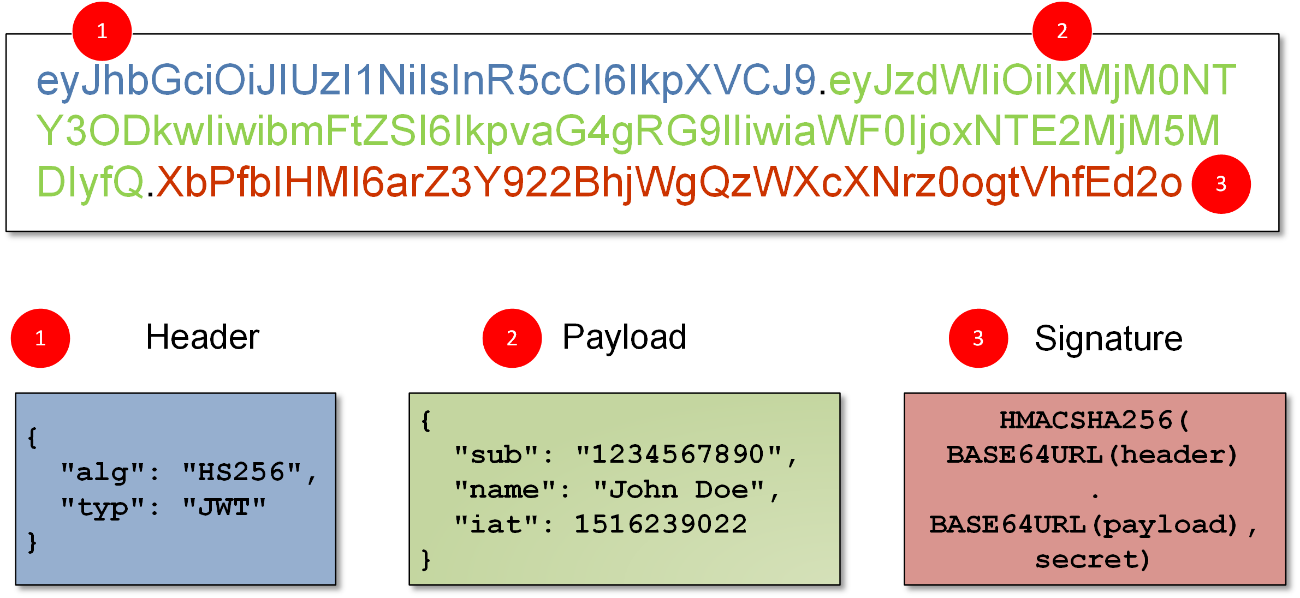
Estos datos se encriptan con un algoritmo HS254 y dan lugar a un fichero cifrado.

La Firma de Verificación está compuesta a su vez por 3 elementos:

* EL header codificado por un algoritmo base 64.
* El Payload codificado por un algoritmo base 64.
* Una cadena secreta que puede estar o no codificada ya que en principio es aleatoria.

Con esto se consigue dos cosas:

* Es necesario conocer la cadena secreta para que un JWT sea válido. Si alguien intenta enviar un JWT con una clave secreta no válida obtendría un fallo del sistema.
* Cualquier cambio en los datos del header o payload darían como resultado una firma de verificación no válida que esta cambiaría.



**Ilustración 10** - JWT structure

1. **¿Qué es la autorización, Roles?**

La autenticación es el proceso de verificar la identidad. Esta identidad puede ser una cuenta de usuario utilizada por una persona, una ID de host utilizada por una máquina, un certificado de servidor utilizado por un componente de software de servidor o un certificado de cliente utilizado por un componente de software de cliente.

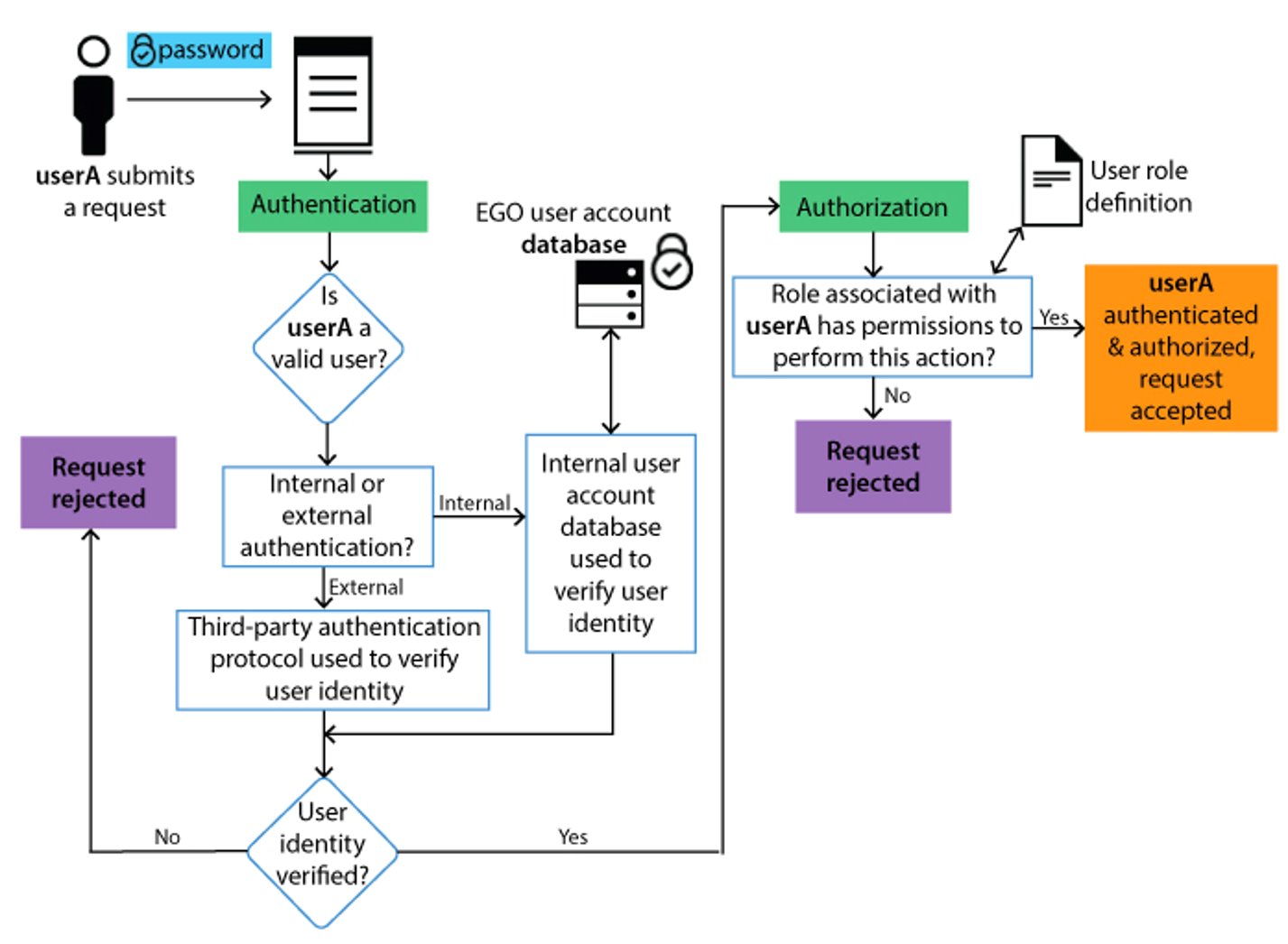
La autenticación generalmente se realiza demostrando que el portador de la identidad tiene un secreto que solo el portador conoce.

Después de autenticar una identidad, la autorización es el proceso de determinar quién puede hacer qué. La autorización se logra mediante la asignación de permisos o funciones a una identidad que accede a los objetos del sistema.

Un permiso otorga acceso a uno o más objetos del sistema. Un rol es un grupo de permisos.

Los roles se pueden asignar a cualquier usuario o grupo de usuarios, y un usuario o grupo de usuarios puede tener más de un rol.

A diferencia de los usuarios jerárquicos, un rol no contiene otro rol.



**Fuente:** [**https://www.ibm.com/docs/en/spectrum-conductor/2.4.0?topic=security-user-roles-accounts-permissions**](https://www.ibm.com/docs/en/spectrum-conductor/2.4.0?topic=security-user-roles-accounts-permissions)

1. **Ejemplo de Protección de API REST con JWT FrontEnd y Backend**

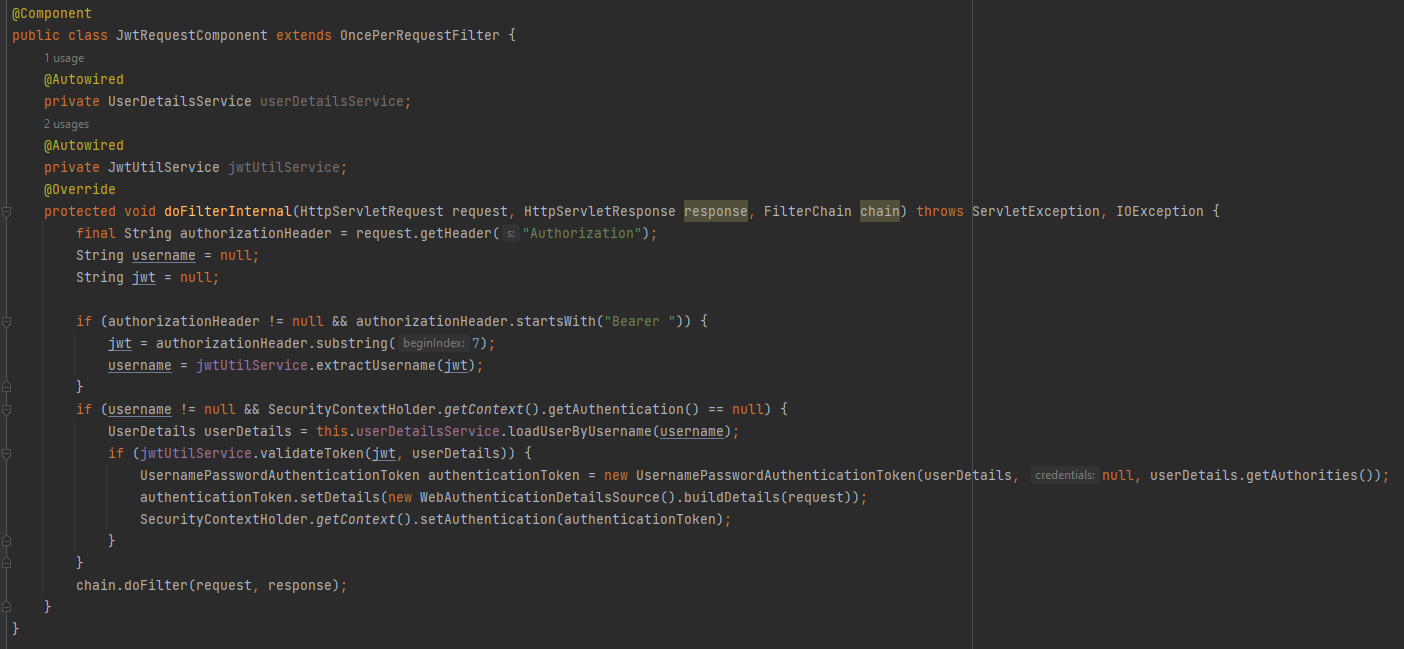
Pom.xml

Config

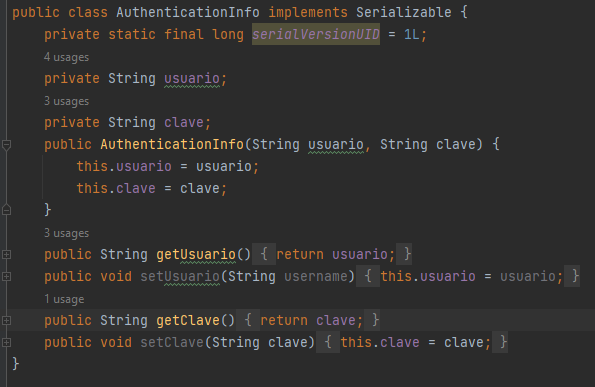


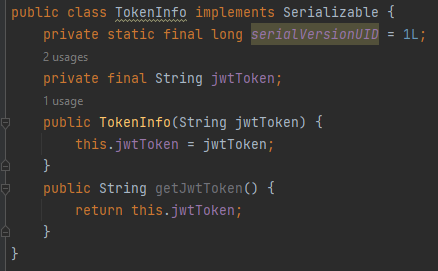


Component

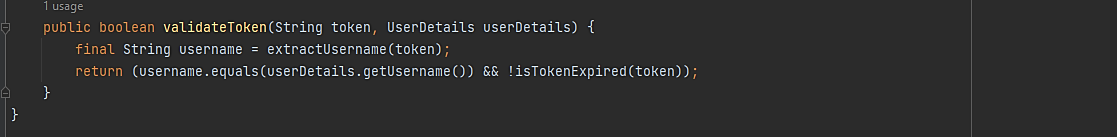
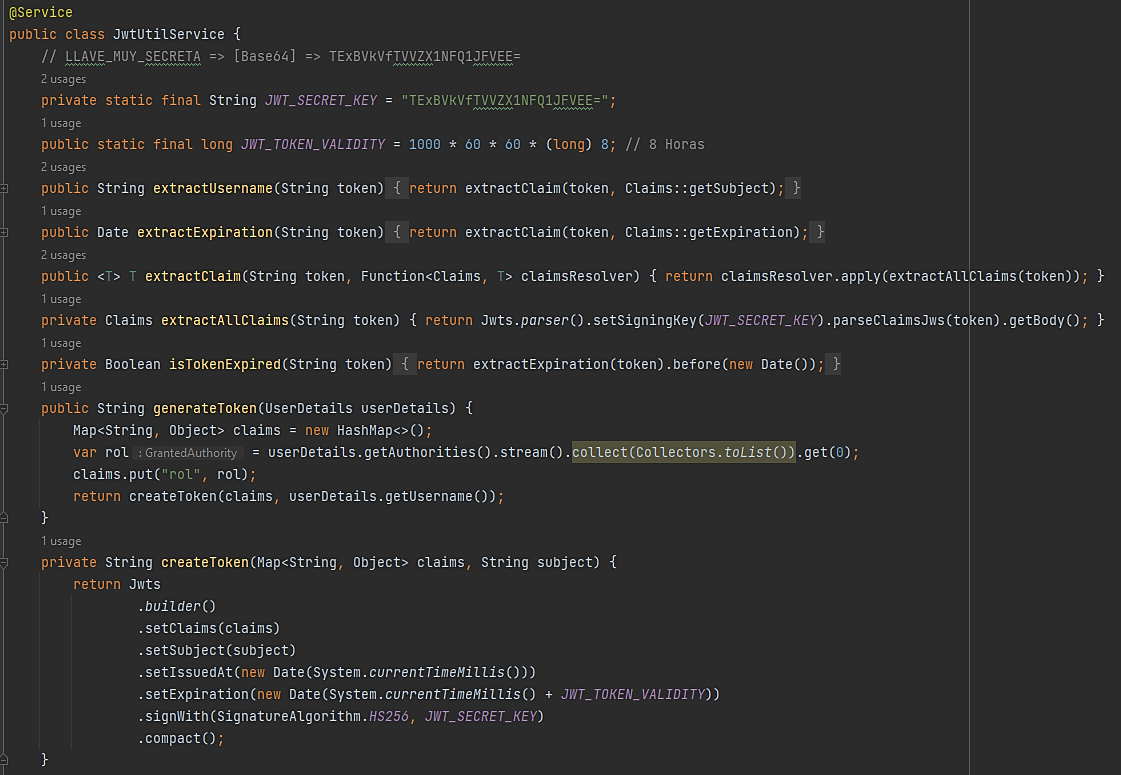


entities



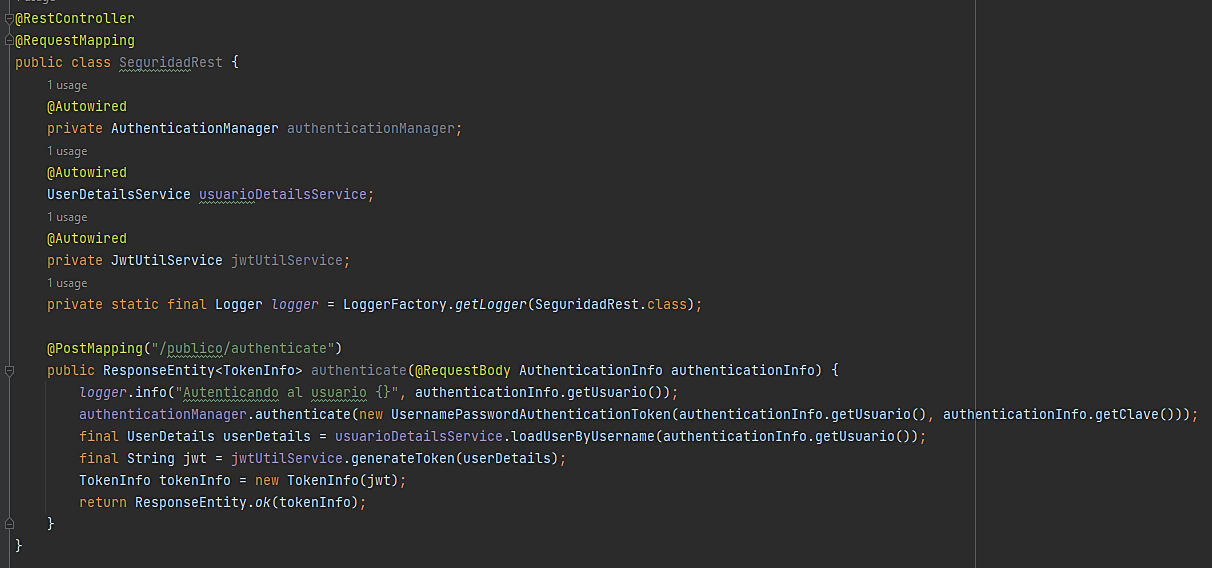


Service

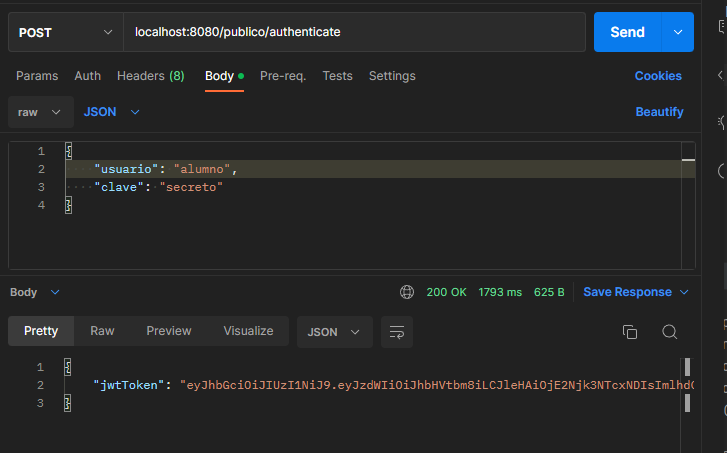


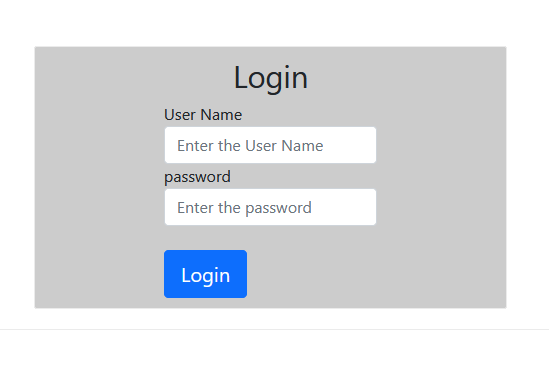


Rest



1. **Pruebas con Postman y Angular Frontend**





1. **¿Qué es lo que más les ha impresionado o gustado?**

* Me ha gustado saber que hay muchos enfoques para integrar la autenticación y la autorización en una aplicación.
* Me ha gustado saber para evitar que en cada llamada el usuario deba entregar su clave y password se usa un token.
* Me ha gustado saber que existen frameworks estándar de seguridad que limiten y registren el flujo de datos de esta forma los proveedores de servicios pueden ser identificados en casos que usen de forma inapropiada los datos de un usuario.

1. **Conclusiones y Recomendaciones.**

* El token generalmente tiene un tiempo de validez, que se establece en el servidor, al momento de generar el token.
* Uno de los estándares para producir los tokens del lado del servidor es JSON Web Token.
* OAuth 2.0 es un protocolo de autorización y NO un protocolo de autenticación.
* OAuth 2.0 utiliza tokens de acceso. Un token de acceso es un dato que representa la autorización para acceder a los recursos en nombre del usuario final.
* Con OAuth 2.0, el Cliente inicia las solicitudes de acceso.