

Antes de empezar con el concepto de json web token, de lo que es jwt vamos a hablar de la problemática de implementar seguridad.

Tenemos varias alternativas para implementar un sistema de login, un sistema de autenticación y también un sistema de autorización con roles, accesos, permisos.

Y lo mas típico es trabajar con sesiones, con una pagina de login con usuario y contraseña. Por ejemplo cuando estamos desarrollando una aplicación donde tenemos todo lo que son las vistas y también tenemos dentro de la misma aplicación el backend un ejemplo típico SpringMVC trabajando con controladores, lógica de negocios, que serian nuestro servicios, y las vistas. Y las vistas se encargan de renderizar, por ejemplo usando timeleef o JCP

Cuando es una única aplicación que contiene todo contiene las vistas junto con los servicios controladores se utiliza sesiones, es lo mas típico.

Otra alternativa tenemos los tokens, son mucho mas escalables pueden funcionar para trabajar, integrar diferentes aplicaciones, ya no estamos hablando de una sola aplicación que contiene todo sino podríamos tener una aplicación en el servidor con un backend que entrega servicio a distintos clientes que son otras aplicaciones. Un front end con angular con React, con View2 en fin, pero también pueden ser aplicaciones de escritorios, aplicaciones moviles o incluso cualquier petición de otros servidores que realizan consultas a nuestro servidor APIRest, separando completamente nuestra aplicación cliente del apirest, del backend, del servidor.





Entonces considerando que la base fundamental del spring security son los token vamos a utilizar mucho esta pagina en nuestra presentación.

Con el manejo de tokens, dos aplicaciones independientes pueden conversar, integrarse sin inconveniente y estas dos aplicaciones serian nuestro Frontend y el Backend. En este contexto, en este caso es donde se utilizan los jwt, los json web Token, donde el usuario va a enviar un código alfanumerico al servidor y el servidor se encarga de descifrar, decodificar y lo valida y va a ver si el usuario existe, si esta registrado en nuestro sistema y también si tiene permiso atravez de sus roles y de acuerdo a esto si puede acceder a determinadas paginas o partes, recursos de nuestra aplicación.

Esto nos permite evitar tener que guardar sesiones al servidor y esto también hace que nuestro servidor funcione mas ligero, bastante mas rápido.

Entrando mas a detalle a que es un Json Web Token, que se abrevia como JWT, es un estándar abierto para implementar seguridad en nuestras aplicaciones ApiRest basado en token el estándar RFC 7519 una forma para representar de forma segura reclamaciones que envia un cliente a un servidor y el servidor interpreta estas reclamaciones o claims que contiene datos, contiene información del usuario como el username. Obviamente no se debe de contener información sensitiva como contraseñas.

Este estandar permite entonces decodificar este código, verificar si es valido y también se encarga de generar este código atraves de una palabra código secreto que tengamos configurado en nuestro servidor. Este código secreto o certificado con la llave publica y privada, que debe de mantenerse en el servidor pues es privado y nunca un cliente debe de obtener esa información ya que con ese código secreto es como se genera el JWT y también como se valida como se verifica, nos permite también ver si es valido, si fue modificado o manipulado.



Algunas de las características importantes:

Un Json web token es muy compacto, debido a su pequeño tamaño en caracteres los json web token se pueden enviar atraves de una petición web, ya sea una URL, un parámetro post, o como se hace típicamente y como es mas recomendable dentro de las cabeceras http authorization, y por supuesto que esto permite que al ser pequeño la transmisión es mucho mas ágil y liviano.

Es completamente autónomo, contiene toda la información sobre el usuario evitando realizar consultas en el servidor, simplemente el jwt contiene toda la información necesaria, no se requiere guardar nada en el servidor, ningún tipo de sesión ni tampoco realizar consultas, esto también mejora bastante la estabilidad de nuestra aplicación.

Respecto a la codificación y seguridad de este código, este json web token hay que tener cierto cuidado, auque el token esta firmado para asegurarse que es autentico no esta encriptado en una sola via sino que esta codificado en doble via por lo tanto es reversible, se puede leer esta codificación, se puede decodificar, por lo tanto una persona con un poco de conocimiento puede leer esta información y también la puede manipular.

Mas adelante vamos a ver un ejemplo de como atraves de código sencillo y en el propio navegador sin necesidad de venir a la pagina jwt se puede ver el contenido de un token…

Pero igual… después vamos a ver que es completamente seguro ya que cuando se envia el json web token el servidor verifica la firma y va a saber realmente si fue manipulado o no. Lo importante es que se puede leer, se puede manipular, entonces no podemos tener ni contraseñas ni mucho menos guardar datos de tarjetas de crédito o cualquier otra información sensible, esto nunca se debe de hacer.



**Tiempo de vida de un token**

Respecto al tiempo por defecto sino se define un tiempo de caducidad donde caduque este Json web token va a ser por tiempo ilimitado, por eso por buena practica debe de establecerse un tiempo de caducidad.

Y acá hay un tema importante, supongamos que dimos mucho tiempo de vida a nuestro token, un tiempo de caducidad bastante amplia, dos semanas o tres semanas por ejemplo, y necesitamos deshabilitar la cuenta de ese usuario, la podemos deshabilitar via base de datos, pero como el usuario conserva una fecha amplia de vencimiento en el token el usuario va a poder continuar accediendo nuestra aplicación porque el token es valido y no ha caducado, por eso es que hay que planificar muy bien respecto a cuanto tiempo de vida debe de tener nuestro token. Por lo mismo no se recomienda dar mucho tiempo de caducidad, y tampoco se recomienda dar muy poco tiempo, por ejemplo si damos 30 minutos o 10 minutos vamos a obligar al usuario a iniciar sesión a cada rato, por lo tanto la recomendación es analizar bien el contexto.

Y precisamente esta es una de las grandes diferencias con las sesiones, ya que las sesiones están del lado del servidor, la desactivación del usuario se hacen de forma inmediata, por ejemplo desactivamos la cuenta del usuario y de forma automática también podemos invalidar la sesión ya que esta dentro del propio servidor, y por supuesto el método de autenticación de sesiones es siempre staifull, es con estado. Mientras que con el token es stailess, no tiene estado, no mantiene una persistencia es decir que cada vez que necesitamos acceder a un recurso protegido en el servidor debemos de enviar nuestro token.



Veamos como funciona nuestra autenticación con Json Web Token.

Aca tenemos dos contextos,

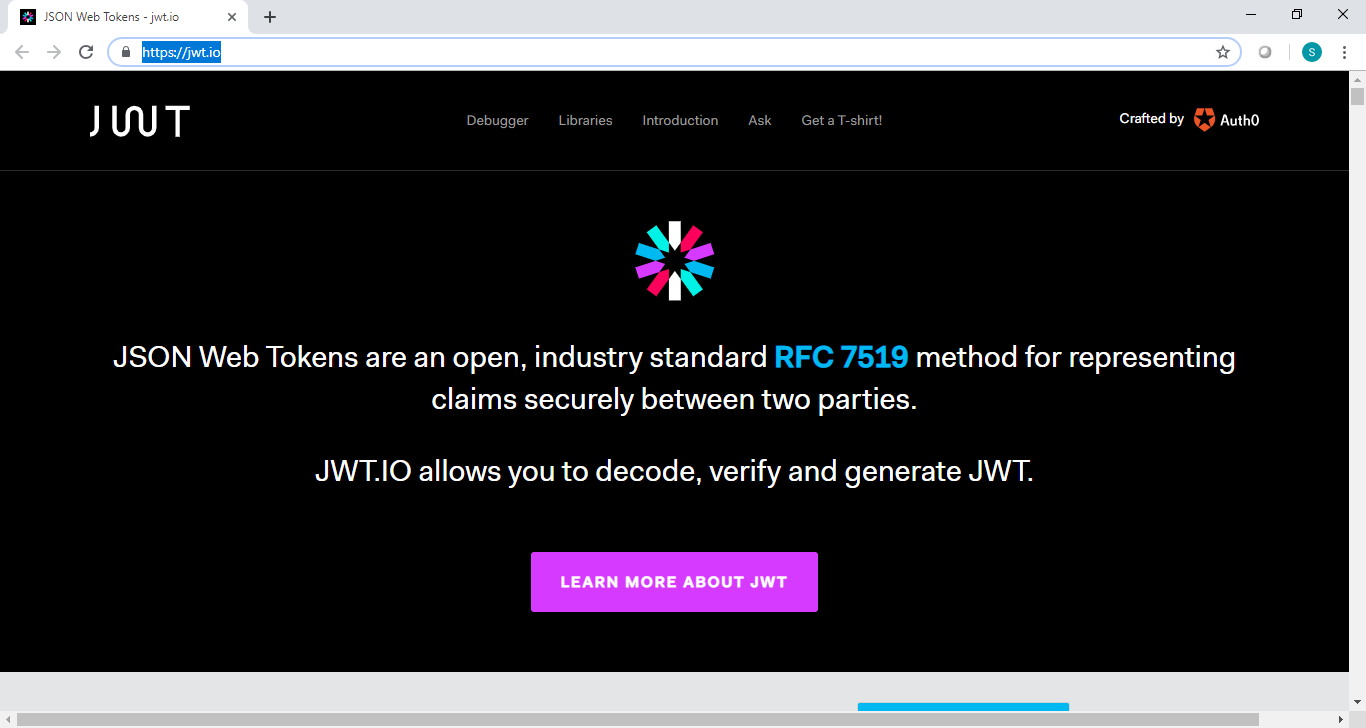
* primero tenemos el cliente que sería nuestro frontend, que puede ser Android, angular, etc.
* Y aca tenemos nuestro apirest, el servidor
* Entonces un cliente se conecta y realiza inicio de sesión, un login, una petición post a nuestra URL Api Login enviando el username y password al servidor
* El servidor valida, en nuestro caso usando Spring Security atraves del Authorization server del API OAuth2
* Y luego tenemos un nodo de decisión, si falla entonces recibimos un código 401
* Pero si la validación es valida entonces crea un token un Json Web Token con un secret
* Retorna el jwt al cliente y este el cliente lo guarda en el local storage o en el sesión storage, si se guarda en el local storage quedara de forma persistente en nuestro equipo, incluso si reiniciamos el equipo siempre va a estar guardado nuestro JWT, mientras que si se guarda en el sesión storage va a quedar almacenado mientras este activo nuestro navegador, si cerramos el navegador automáticamente va a desaparecer el json web token y después tendremos que volver a iniciar sesión.

Esta seria la parte de autententicacion, ya tenemos nuestro json web token, lo tenemos almacenado y lo podemos utilizar. Por lo tanto ya podemos navegar en la aplicación, podemos acceder a paginas que requieren acceso, que están protegidas.



Ahora una vez autenticados, veamos como funciona cuando ya contamos con el token.

* El cliente hace una petición al servidor, accediendo a los recursos en el servidor, enviando una petición http del tipo get, post, put o delete, cualquiera. Y lo relevante de esta petición es de que como ya contamos con el token este deberá ser enviado en la cabecera del http, y este se envia dentro del authorization, en el header HTTP.
* El servidor recibe el token y verifica la firma del Json Web Token, va a validar que no haya sido manipulado, que este correcto
* y si paso algo extraño, si el token no es valido da mensaje acceso denegado, 403 Forbiden, prohibido no lo deja continuar para acceder al recurso.
* Si esta todo bien, es un json valido, esta correcto. Obtiene los datos del cliente, del usuario, los Claims, se le conoce como reclamaciones que contiene el username y los roles, … luego realiza la autenticación, verifica los permisos y roles y pasa al nodo de decisión.
* Si el usuario no tiene permisos al recursos, entonces acceso denegado, código 403 Forbidden
* De lo contrario, si esta autenticado correctamente, tiene las autorizaciones correspondientes, entonces se realiza la consulta a la base de datos atraves del servicio APIRest y retorna la respuesta al cliente, ya sea un Json o XML.



Ahora vamos a la pagina de Json Web Token.io y aca vamos a hacer un ejercicio simple para entender un poco mas de este estándar.

Lo primero es entender que esta compuesto por 3 partes, que están separadas por un punto.

Si se fijan hay tres colores,

El color rojo

Un color morado

Y un color celeste.

Estas tres partes conforman el json web token.

El primero el Header en color rojo, la cabecera que contiene información sobre el algoritmo el cual se va a codificar nuestro token y también el tipo.

En este caso es un algoritmo HS256 y un typ tipo JWT

Luego tenemos el punto como separador

Y luego el Payload que serian los datos o la información del usuario

Primero como estándar tenemos el sub que representa un identificador, típicamente es el nombre de usuario, el username con el cual el usuario se identifica en nuestra aplicación, puede ser el SOEID por ejemplo, pero también puede contener otros datos como el nombre, el apellido, y cualquier otra información adicional que deseemos colocar sin exponer al usuario, es decir sin mostrar data sensitiva.

Posee una fecha de creación y también opcionalmente se puede colocar una fecha de expiración, y también podemos tener información sobre los permisos, los privilegios o roles del usuario.

Lo que tenemos que considerar es que es totalmente reversible, un token se puede leer, alguien que sepa un poco de programación puede poder ver el contenido de un token. Ya haremos un ejemplo de como con un poco de código esto se puede mostrar.

Finalmente tenemos la parte de abajo, que verifica la firma. Para esto necesitamos un código secreto, por ejemplo un código secreto cualquiera y este se debe de almacenar en el aplicativo backend, de acuerdo a el código secreto se genera el Json Web Token, y nos sirve para verificar la validez de un token. Recordando entonces el cliente no puede tener acceso al código secreto, solamente se maneja del lado del servidor y esto es lo que hace seguro el Json web token, la verificación de firma, la cual sirve para verificar que el token no haya sido manipulado. Y si lo fue entonces podrá detectarse y podrá también marcarse o informar de que no es un token valido.

Y por debajo entonces de forma interna la tecnología JWT unifica las tres partes y lo codifica en base 64, todo esto codificado con la palabra secreta de forma segura.

(ahora modificar el json web token para que de firma invalida)

(ahora modificar el json web token en el payload para agregar datos)

Ahora respecto al almacenamiento, un Json web token puede almacenarse en el local storage o en el sesión storage ( con un f12 entramos a ver la ubicación del local storage )

Local Storage: Recodemos que la información permanece de forma persistente en el equipo, por lo tanto podemos cerrar el navegador, podemos reiniciar la maquina y la información continuara almacenada en el equipo.

Session Storage: La información permanece en el navegador hasta que lo cerremos.

Entonces cuando usar uno y cuando usar otro…

Por ejemplo si queremos guardar el token con un tiempo de caducidad extensa o indefinida entonces usamos el local storage, que dure semanas o un tiempo ilimitado.

Pero si queremos que solo este disponible un par de horas podemos usar el sesión storage, y también depende del tiempo de expiración, ya que si queremos que nuestro json web token dure 7 dias, una semana guardamos el token en el local storage.

Ahora aparte del almacenamiento, nuestras implementación debe de asegurase que los token no hayan sido manipulados y sean validos, atraves de la firma. Y en caso de que asi fuera detectar el cambio, invalidad el token.

Ok, ahora con F12 entremos a la consola y hagamos un ejercicio y veamos como puede extraerse la información de un token con unas cuantas líneas de código…

let token = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIyNjA0IDA0NTE5IDAxMDEiLCJuYW1lIjoiRnJlZHkgU2ljYSIsImVtYWlsIjoiZnJlZHlzaWNhQHlhaG9vLmNvbSIsImlhdCI6MTUxNjIzOTAyMn0.q956VSY77TfzNk1VCJUjTydsNN4-rOB4Tggbkra2Bt0"

primero vamos a crear una variable con java script…

let payload = token.split(".")[1]

Luego vamos a extraer el payload

payload

"eyJzdWIiOiIyNjA0IDA0NTE5IDAxMDEiLCJuYW1lIjoiRnJlZHkgU2ljYSIsImVtYWlsIjoiZnJlZHlzaWNhQHlhaG9vLmNvbSIsImlhdCI6MTUxNjIzOTAyMn0"

window.atob(payload)

ahora vamos a decodificar la variable payload con base 64, simplemente usando el objeto global de java script WINDOW punto el método atob y pasamos el payload

"{"sub":"2604 04519 0101","name":"Fredy Sica","email":"fredysica@yahoo.com","iat":1516239022}"

Con un JSON.parse podemos decodificar por completo

JSON.parse(window.atob(payload))

Y como vieron no es recomendable enviar información sensible ya que puede ser visible este tipo de información.

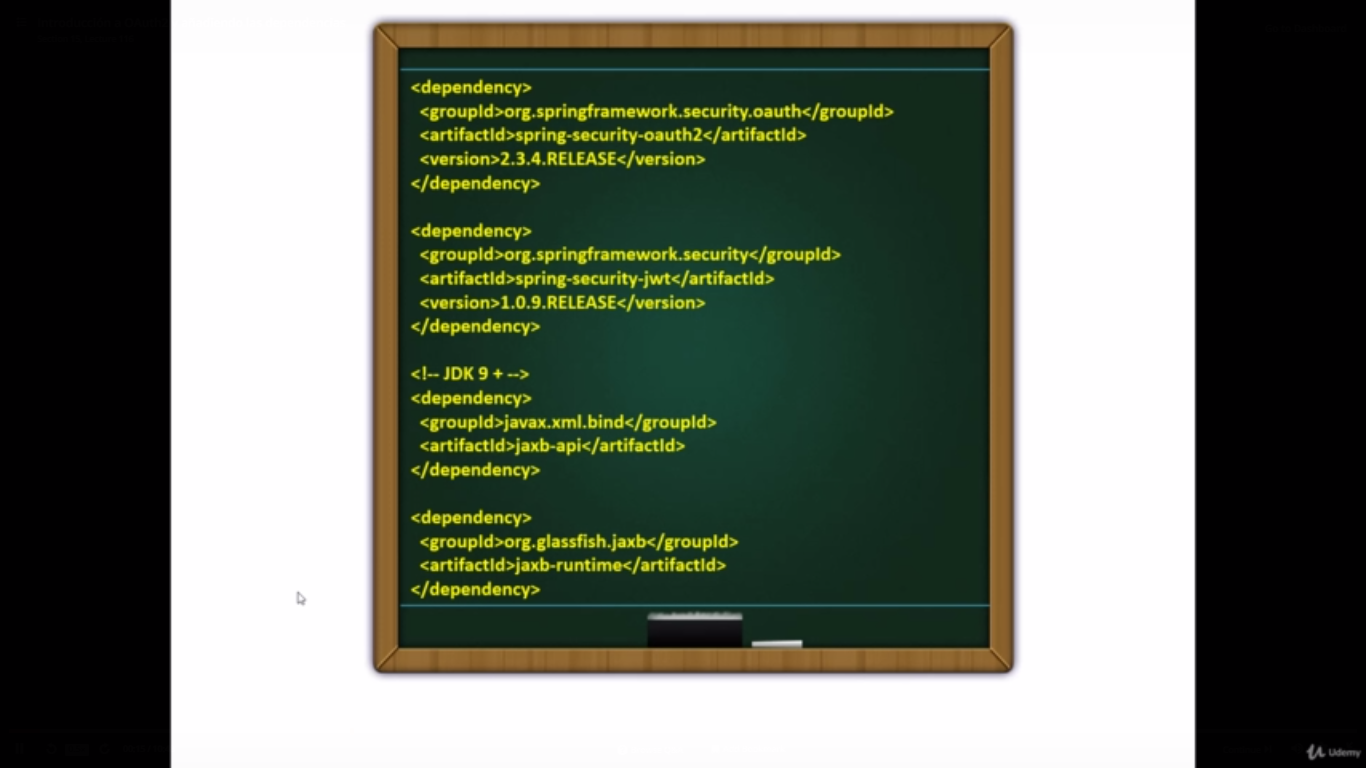
Ok….Ahora si continuamos viendo la pagina veremos el listado de librerias…

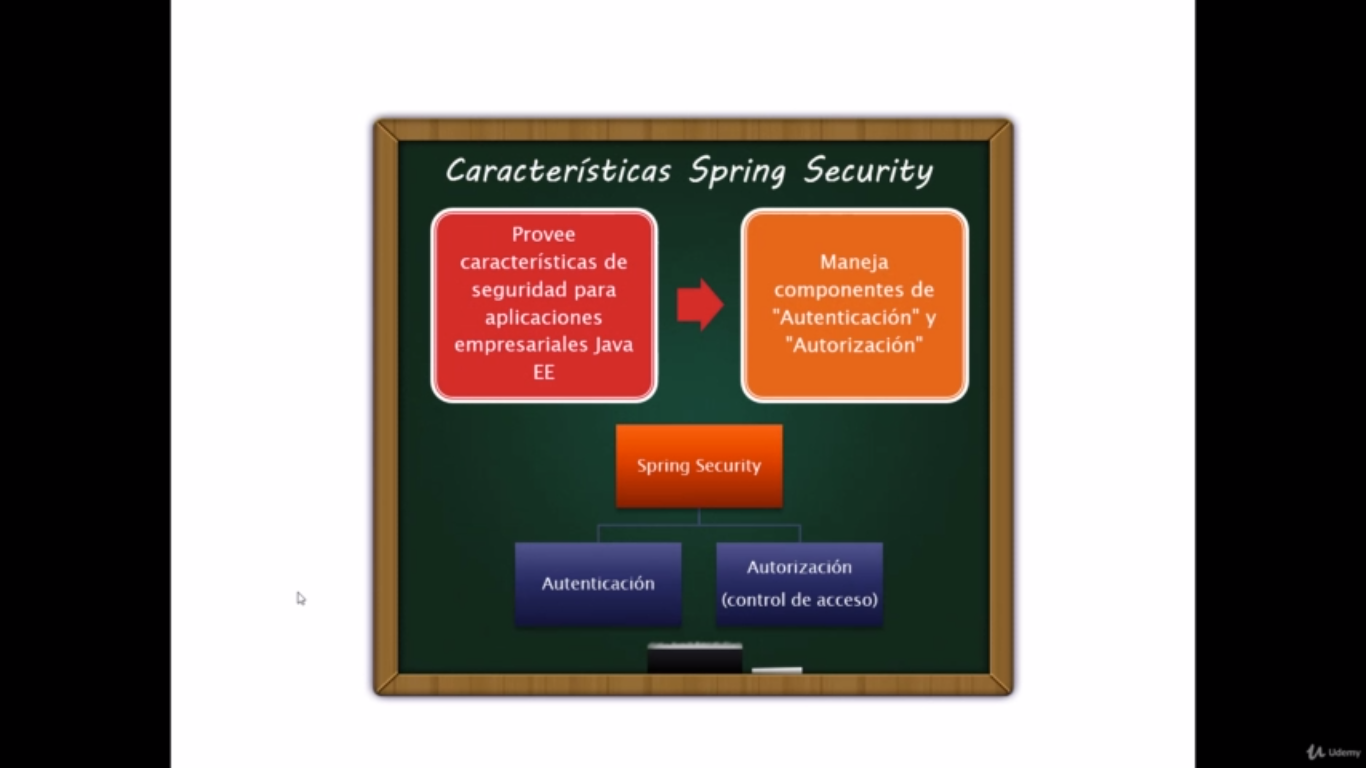
Pero en nuestro caso y derivado del proyecto de Eclipse, todo nuestro backend iba a estar basado en la tecnología de Spring, entonces vamos a la pagina <https://spring.io>, si vamos a los proyecto

<https://spring.io/projects/spring-security-oauth> tenemos la implementacion para trabajar con el estándar OAuth2, trabajando con Json Web Token. Nos vamos a la documentación donde aparece la versión y nos aparecen las librerias, las dependencias de MAVEN instrucciones, el group ID y el artifact.

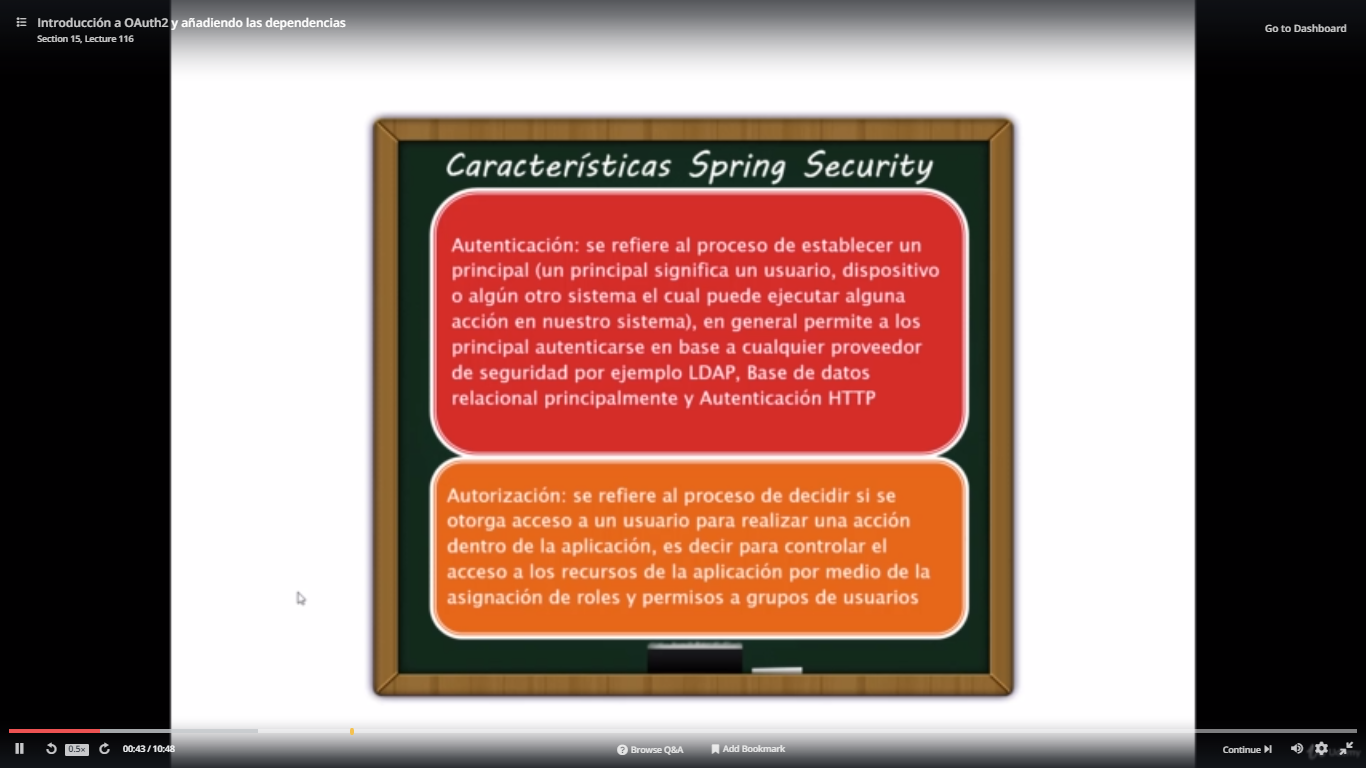


Ok, ahora ya entraremos de lleno a lo que es Spring Security y su implementación OAuth2 usando Json Web Token, JWT, lo primero que haremos es instalar y configurar la dependencia en nuestro POM XML, las librerias.

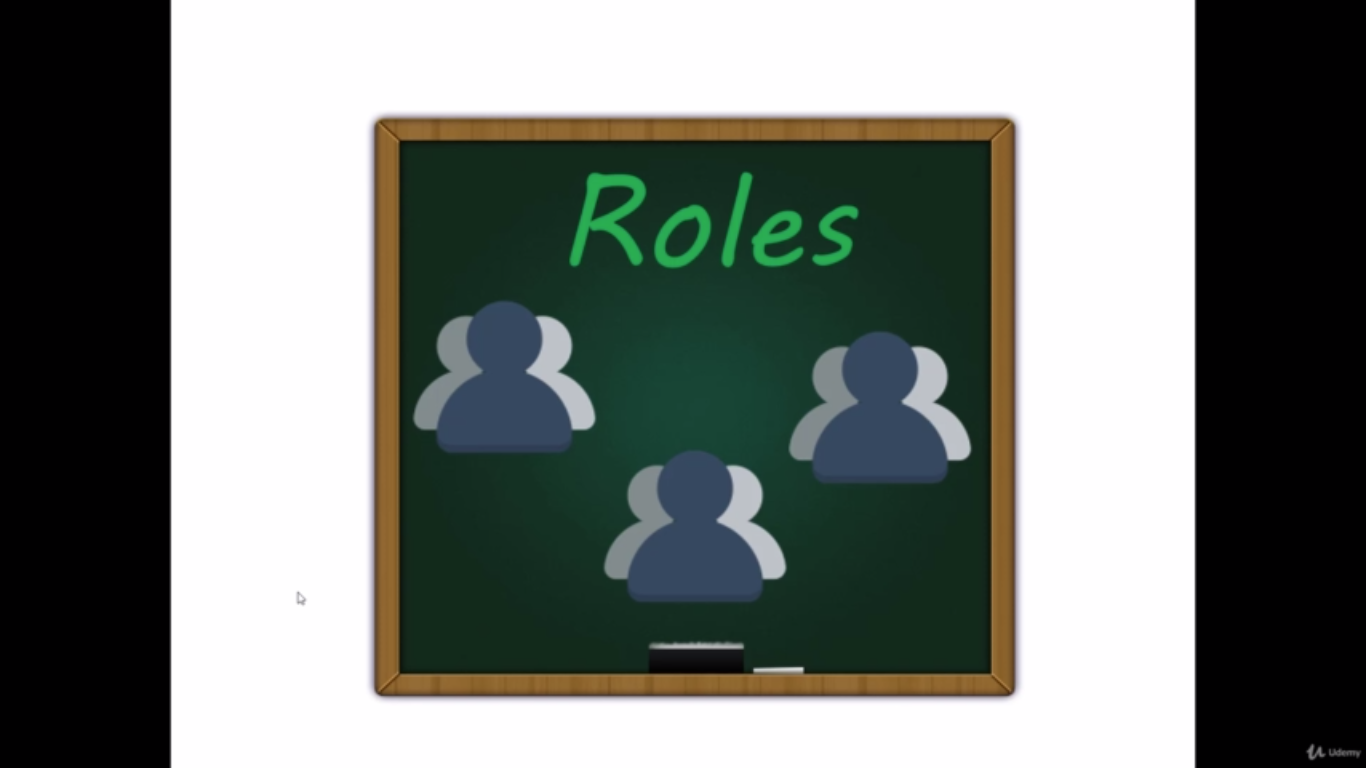




Y estando aca, cabe la pena señalar que Spring Security es un framework de seguridad. es un proyecto que se viene a integrar a SpringFramework. Ofrece lo que es autenticación, es decir lo que es Login de usuario, atraves de credeciales, username y password y también todo lo que es autorización.

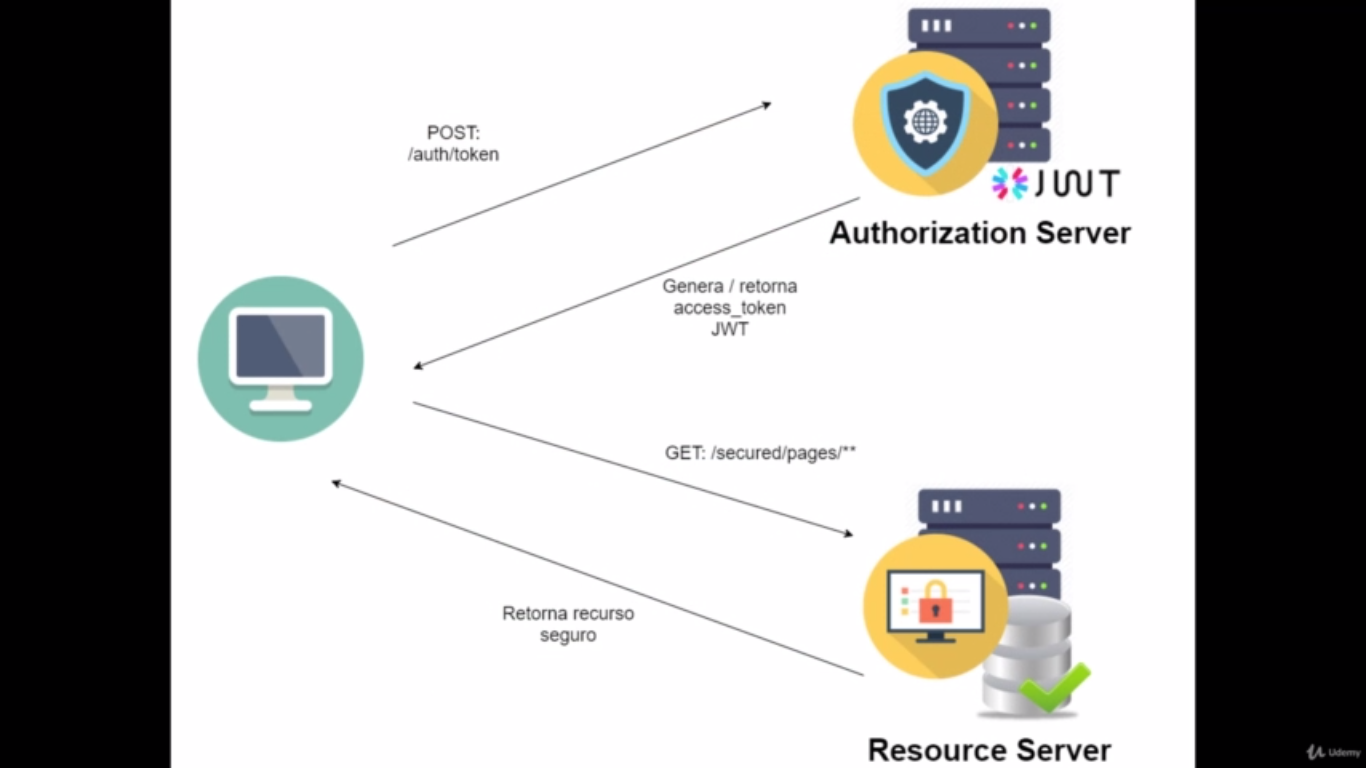


Cuando se refiere a autorización, se refiere a ACL que significa básicamente una lista de control de acceso, es una forma de determinar los permisos de accesos apropiados a un determinado recurso, por lo tanto nuestro recurso va a ser el backend, nuestro apirest, nuestros controladores, también pueden ser clases de servicios, lógica de negocios y al usuario le damos acceso a estos recursos atraves de roles.



Y un rol es un grupo o tipo de usuario, al que se le otorga ciertos privilegios para llevar a cabo una o varias acciones dentro de nuestra aplicación, son construcciones simples que constan de un nombre por ejemplo: usradmin, admin, usuario, moderador, etc.

Puede consederce ya sea a los usuarios o incluso a otros roles, se utiliza para crear grupos lógicos de usuarios para la asignación adecuada de los privilegios en nuestra aplicación, considerando que los recursos son las paginas de nuestra aplicación.



Entonces, la implementación que se iba a realizar en el proyecto de Eclipse era con Oauth2 de Spring, es un framework, es un protocolo estándar de autorización, que permite a las aplicaciones de terceros, ya sea Angular, React, Android autenticarse con cuentas de usuario que están en otra aplicación en un servicio HTTP externo. Basicamente para compartir un sistema de autenticación de una aplicación a otra, sin compartir toda la información del usuario, ni menos las credenciales. Este mecanismo es utilizado por compañías como Google, como Facebook, como twitter o como GitHub, para permitir a los usuarios compartir información sobre sus cuentas de forma limitada con aplicaciones de terceros, independiente de la plataforma, si son aplicaciones moviles o aplicaciones web, ya sea con Angular, con React, con Android… con cualquier tecnología.

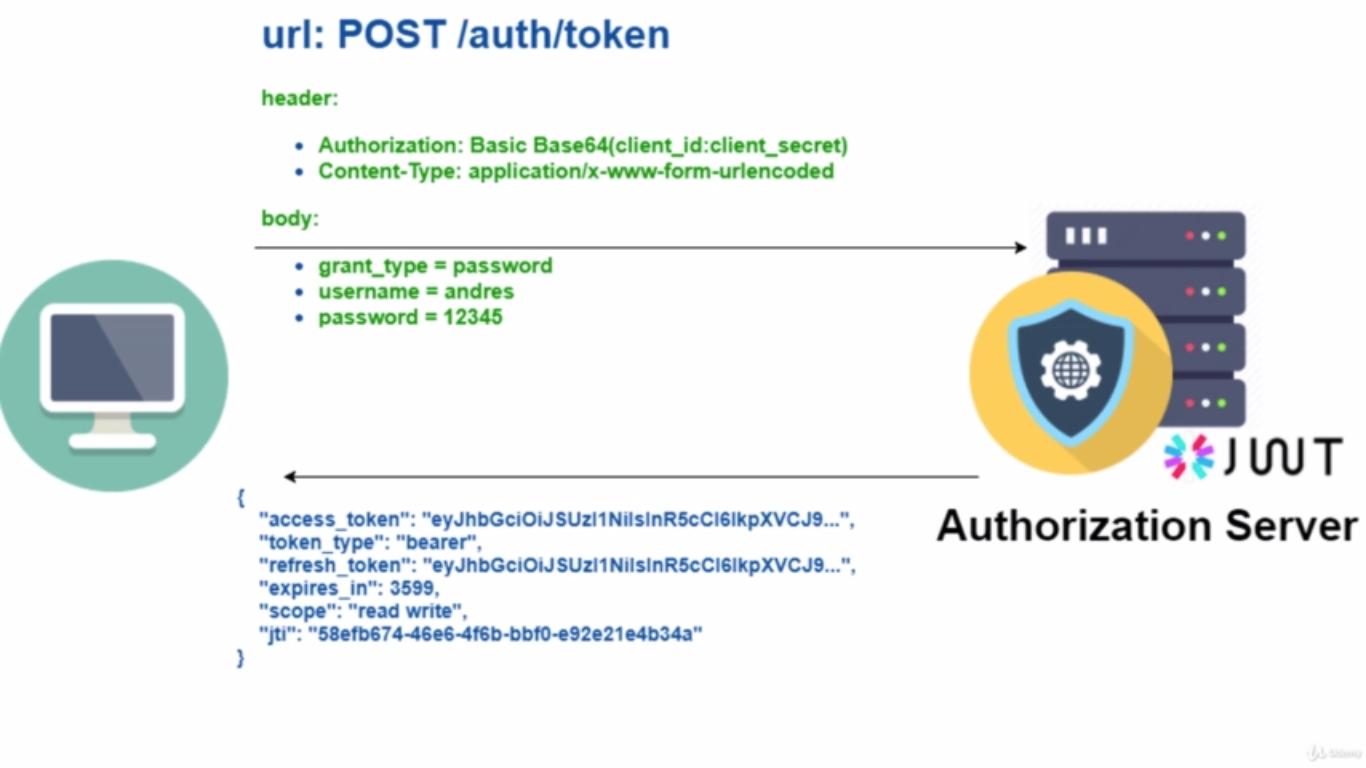
La implemetacion que trae Spring security OAuth2 se compone de dos partes:

Primero tenemos el servidor de autorizaciones, el authorization server.

Por otro tenemos el Resource Server o servidor de recursos.

El primero se encarga de realizar la autenticación del usuario, y si es un usuario valido genera el token de acceso y se lo retorna al usuario, retorna el token de acceso para que después el usuario con este token pueda acceder a los distintos recursos de nuestra aplicación.

Por otro lado tenemos el servidor de recurso que se encarga de administrar los permisos y accesos hacia nuestras paginas APIRest y EndPoints que están con seguridad. Entonces por debajo el resource server habilita un filtro de spring OAUTH2AUTHENTICATIONPROCESSINGFILTER el cual se utiliza para validar y dar acceso a una petición HTTP o Request que viene desde un cliente, este request es el que envia el token de acceso en las cabeceras y si es valido el servidor de recurso otorga el permiso para que el cliente pueda acceder a los recursos, paginas o endpoints del backend… básicamente nuestras apirest.

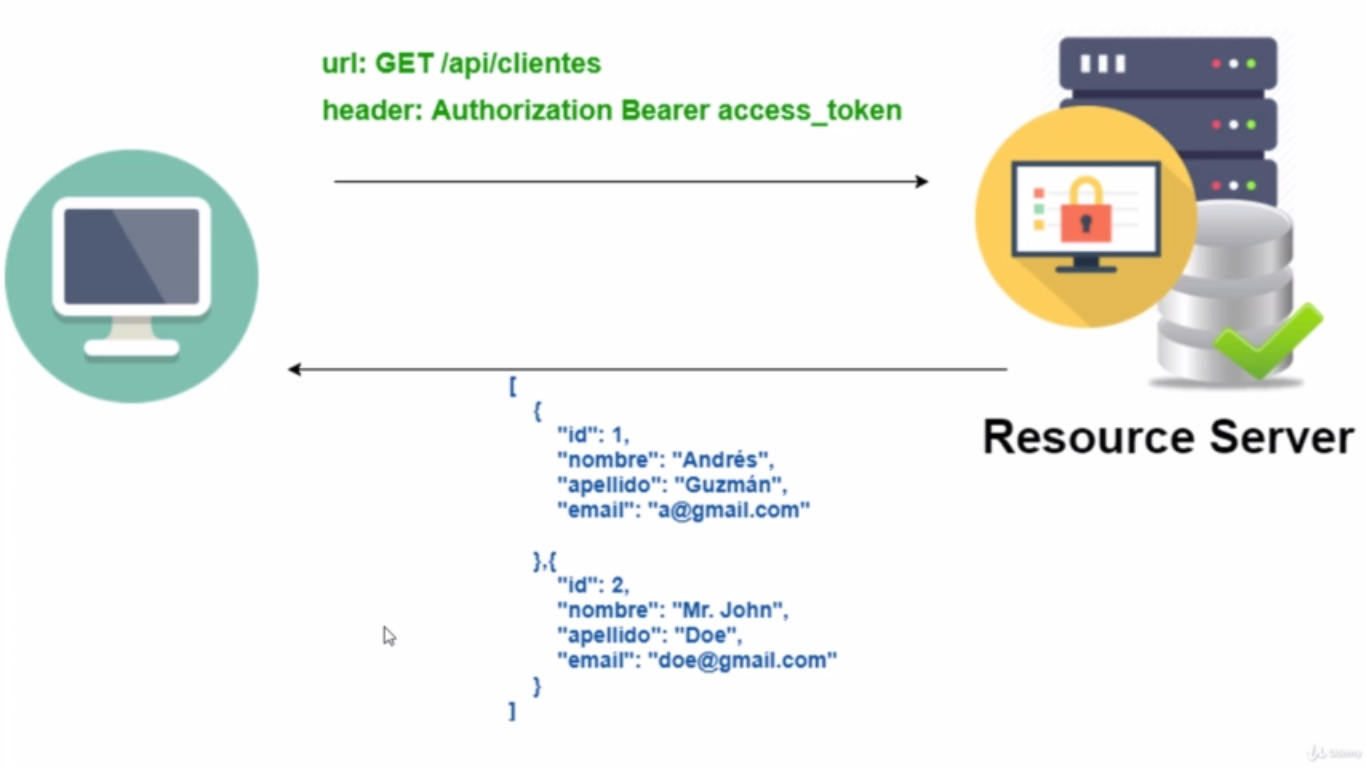


Ahora veamos un poco mas a detalle el proceso del authorization server…

Primero tenemos una ruta URL del tipo POST Auth2, esta ruta es proveida por spring y nos permite realizar el login, la autenticación y por debajo si todo esta bien va a generar el token y para ello nosotros tenemos que enviar en el EndPoint en la ruta Auth/Token tenemos que enviar algunos paramatros que son importantes como en el Header:

El parámetro Authorization: contiene el client\_id y el client\_secret que son dos parámetros del API OAuth2 lo tenemos que concatenar con los dos puntos…y encriptar en base 64. Una autorización del tipo basic…. Y ademas el content type del tipo Application/x-www-form-urlencodeded…y en el cuerpo del request tenemos que enviar las credenciales, el username, el password y el grant\_type del tipo password.

Estos parámetros se envían al servidor, el servidor valida y si todo esta bien va a generar el token de acceso con algunos parámetros mas…



Ok, ahora veamos el servidor de recursos…

Basicamente si queremos acceder a una pagina protegida de nuestro backend… de nuestra aplicación… aca tenemos por ejemplo la ruta del tipo get api clientes y en la cabecera tenemos que enviar el Authorization…el atributo Authorization del tipo bearer… con nuestro token de acceso.

Aca entonces iría el token que recibimos cuando iniciamos sesión, si el token es valido otorga los permisos y accesos para ver el recurso. Y va a otorgar por ejemplo el contenido en JSON

**https://spring.io/projects/spring-security-oauth#learn**

Ok, ahora retornamos a la pagina de Spring security y vamos a la documentación en Learn y nos fijamos en la ultima versión… es muy importante que siempre sea la ultima versión estable, es decir la current. Para nuestros desarrollos se recomienda la current que es la estable, no se recomienda la snapshot, ni alfa, ni beta siempre las versiones finales, es decir las actuales.

Ahora vamos a la documentación y encontraremos el link para la dependencia de Maven, hacemos un click en instrucciones…….. y encontraremos la versión 1 y dos.

Entonces ahora copiamos el artifact id de spring 2 y nos vamos al repositorio de Maven

[**https://mvnrepository.com/**](https://mvnrepository.com/)

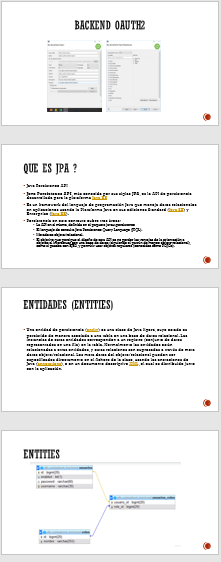
En buscar colocamos spring-security-oauth2 …….

En buscar colocamos spring-security-jwt

Ahora aca hay una advertencia, si estamos utilizando JDK 9 o superior necesitamos agregar dependencias de Jaxb, el api de Jaxb, lo que sucede es que en las versiones anteriores al 9 se incluía JaxBi pero a partir de la versión 9 se excluyo…entonces ahora hay que incluirlo. Sino se incluye podría generar errores al momento de compilar.

Siempre se debe de considerar utilizar las versiones.

En caso de necesitar Jaxb se debe de incluir también la implementación.



# Creando el repositorio JPA – IusuarioDao

Ok, ahora vamos a evaluar el repositorio JPA que creamos con el nombre de IUsuarioDao, esta es una implentacion del Dao para realizar la consultar por el username. Ya veremos como implementar una consulta personalizada para obtener al usuario mediante el username para el login. Aca se utiliza el Api Crud repository que se ha estado utilizando en las anteriores presentaciónes.

# Creando la clase de servicio UsuarioService

Ok, ahora vamos a ver el servicio donde se implementa una interfaz propia de SpringSecurity, USERDETAILSERVICE para obtener el usuario por el username y los roles.

…

# Ver la clase SpringSecurityConfig y registrar el UserDetailService JPA

Ok, ahora para finalizar la implementación de SpringSecurity con JPA veremos la configuración de SpringSecurity respecto al AuthenticationManager, tenemos que registrar la clase de servicio que implementa la interfaz UserDetailService la tenemos que registrar en el AuthenticationManager para que se pueda realizar el proceso de authenticacion utilizando JPA. Todo lo que implementamos…los usuarios con sus roles.

# Ahora veremos la clase para añadir la configuración para el servidor de autorización

Este se encarga del proceso de autorización del lado de OAuth2 que se encarga del proceso de Login, crear el token y validarlo. Evaluamos entonces la clase AuthorizationServerConfig

…

Esta clase tiene tres definiciones de configuración que sobreescribimos, la primera

configure(AuthorizationServerEndpointsConfigurer endpoints), Se encarga de todo el proceso de autenticacion y de validar el token, Cada vez que iniciamos sesion enviamos nuestro usuario y contraseña y si todo sale bien realiza la autenticacion, genera el token, se lo entrega al usuario y con ese token puede acceder a las paginas y recursos de nuestra aplicacion backend pero para eso, para poder acceder se tiene que validar. Y eso se realiza en Endpoints en unas rutas que maneja el servidor de autorizacion. tanto para el login autenticacion donde genera el token y tambien para el proceso de validacion(validar el token y su firma)

configure(ClientDetailsServiceConfigurer clients), aca configuramos nuestros clientes, las aplicaciones que van a acceder a nuestro ApiRest, en este ejemplo tenemos un solo cliente, pero en caso de tener varias aplicaciones que consuman este servicio rest debemos definir cada uno con su código secreto y contraseña.

La idea del estándar es dar mayor seguridad, no solo nos autenticamos con los usuarios sino también la aplicación debe de registrarse con sus credenciales.

configure(AuthorizationServerSecurityConfigurer security), aca se configura los permisos de nuestros EndPoints, nuestras rutas de acceso pero de SpringSecurityOAuth2. Tenemos dos endpoints en el authorization server, uno para autenticarnos, para iniciar sesion y se encarga de generar el token y enviarlo al usuario Y esa ruta tiene que ser publica, ya que todos los usuarios podran iniciar sesion, autenticarse y recibir un token.

Y con estas tres configuraciones completamos para el servidor de autorización.

# Ahora añadiremos la configuración para el servidor de recurso.

Evaluaremos entonces la clase ResourceServerConfig que se encarga de dar acceso a los clientes a los recursos de la aplicación, siempre y cuando el token sea valido.

….

# Ahora evaluamos el insert de los usuarios.

# Ahora evaluamos la clase springbootbackendapirestapplication.

# Ahora probamos en POSTMAN

# <http://localhost:8091/api/clientes>

# <http://localhost:8091/api/clientes/1>

# <http://localhost:8091/oauth/token>

***body***

x-www-form-urluncoded

username andres

password 12345

grant\_type password

***Authorization***

Type Basic Auth

Preview request

Send…

# <http://localhost:8091/api/clientes/1>

***Authorization***

Type Bearer Token

Token $#()UFWHKFJHSLJKFHLKSJ

Preview Request para agregarlo al header

# Mostrar asignación de llaves secretas MAC para firmar un Token JWT

AuthorizationServerConfig

En el código el JwtAccessTokenConverter notamos que tiene la definición de la llave, sino se asigna entonces automáticamente el API va a generar uno por nosotros, ese código se maneja en el servidor y nunca en el cliente.

La tercera es utilizar un código de tipo RSA, este requiere un certificado publico y privado. Se genera el token con la llave privada y con la publica podemos validar y verificar la integridad del token para luego autenticar. Este es un algoritmo mucho mas robusto y seguro.

# Ahora vamos a crear y asignar un certificado RSA para firmar un Token JWT

<https://www.openssl.org/>

Para crear el certificado publico y privado tenemos que usar alguna herramienta, nosotros usaremos OpenSSL o bien la que trae Java el JDK Key Tools, OpenSSH atraves de SSH Key Gen, la verdad es que hay diversas herramientas. Pero vamos instalar OpenSSL…

En la pagina nos vamos a community,

En community nos vamos a binarios

Y en binarios nos vamos a wiki

Y aca podemos descargar los binarios para los diversos sistemas operativos, se recomienda instalar la versión completa ya que esta actualizara la variable path del sistema operativo.

Una vez ya instalado podemos ir al command y ejecutamos el comando

*Openssl genrsa -out jwt.pem*

OpenSSL genrsa con el comando -out indicamos el nombre del archivo donde se va a generar y se va a almacenar el certificado… la llave privada por ejemplo jwt.pem, la extensión es importante es una extensión de certificado digital. Damos ENTER y con esta instrucción generamos la llave privada RSA y luego tenemos que mostrar en la pantalla para poder copiar esta llave privada

*Openssl rsa -in jwt.pem*

Entonces con el mismo comando y cambiando out por in y en lugar de genrsa solo rsa, el nombre del archivo debe de ser el mismo del genrsa. ENTER y nos despliega la llave privada.

*Openssl rsa -in jwt.pem -pubout*

Ahora vamos a mostrar la llave publica, es exactamente igual al anterior y agregamos la bandera - pubout, con esta bandera va a mostrar la llave publica.

Y esto es todo lo que necesitamos… y procedemos a copiar.

…

Ok, ahora estamos listos para que nuestras llaves secretas sean atraves de algoritmos RSA, el cual es un sistema criptografico de llave publica y privada para cifrar como para firmar digitalmente.

Vamos a ejecutar…

Con la generación del token no debe haber ninguna variación el proceso es el mismo vamos a generar token y luego lo vamos a utilizar…

# Añadiendo información adicional al Token

Entonces vamos a verificar la clase infoAdicionalToken el cual va a implementar la interfaz TokenEnhancer que es un token potenciador que nos permite agregar más información…