Autenticación y autorización JWT

Desarrollo de Sistemas Web - Back End

Versión 1.0 F. Act 27/09/2024

Autenticación y autorización

Autenticación es el proceso de verificación de la identidad de un individuo o servicio, mientras que autorización es el proceso de evaluar los permisos necesarios para acceder a un recurso específico o realizar una acción específica.

Mientras que la autenticación se enfoca en determinar que el usuario es quien dice ser, la autorización se encarga de controlar qué acciones ese usuario puede realizar.

En otras palabras, la autenticación se utiliza para iniciar sesión en un sistema y la autorización se utiliza para determinar qué acciones puede realizar un usuario una vez que se ha autenticado.

Autenticación

Es el proceso mediante el cual se verifica la identidad de una persona que intenta acceder a un sistema. Esto se logra proporcionando credenciales, como un nombre de usuario y contraseña.

Una vez que el sistema recibe estas credenciales, determina si la persona es quien afirma ser. La forma más común de autenticación es a través de una página de inicio de sesión, pero también existen métodos más avanzados y seguros que utilizan varios factores de verificación.

Por ejemplo, es común recibir un código de verificación enviado a través de un mensaje de texto o correo electrónico para confirmar la identidad del usuario. por ejemplo, en una aplicación como Facebook y en cómo maneja la autenticación.

Cuando iniciamos sesión con nuestro correo electrónico y clave, Facebook está autenticando, es decir, está comprobando que la contraseña ingresada coincide con la contraseña que se asoció a nuestro correo electrónico. En tanto la contraseña coincida, podremos acceder a la aplicación. Pero si no coincide, Facebook no nos permitirá iniciar sesión porque no puede asegurar que somos quienes decimos ser.

Métodos de autenticación más utilizados

1. Autenticación Multifactor (MFA)

- Combina múltiples métodos de verificación, como algo que el usuario sabe (contraseña), algo que tiene (código SMS o aplicación de autenticación), o algo que es (huella dactilar).
- Proporciona una capa adicional de seguridad, ya que un atacante necesitaría comprometer más de un factor para acceder.

2. Sin Contraseña (Passwordless)

- Métodos como el "enlace mágico" que envía un enlace de acceso al correo electrónico del usuario.
- Facilita el acceso sin necesidad de recordar contraseñas, pero depende de la seguridad del correo electrónico del usuario.

3. Autenticación por Redes Sociales

- Permite a los usuarios iniciar sesión utilizando cuentas de redes sociales como Facebook, Google o Twitter.
- Agiliza el proceso de registro y acceso, pero puede implicar compartir información de perfil con el servicio.

4. Autenticación API

- Certifica la identidad del usuario al acceder a recursos en el servidor mediante protocolos como OAuth.
- Incluye métodos como autenticación básica HTTP y autenticación basada en tokens, que son fundamentales para APIs RESTful.

5. Autenticación Biométrica

- Utiliza características físicas del usuario, como huellas dactilares, reconocimiento facial o escaneo del iris.
- Cada vez más común en dispositivos móviles, mejora la seguridad al requerir datos que son únicos para el usuario.

Autorización y Seguridad del Sistema

La autorización es el proceso de evaluar los permisos necesarios para acceder a un recurso específico o realizar una acción específica. Es lo que define a qué recursos de sistema el usuario autenticado podrá acceder.

Seguridad del Sistema: Es el resultado de la combinación de autenticación y autorización, por lo que es importante considerar ambos aspectos al desarrollar un proyecto web. Ambos procesos de seguridad proveen capas adicionales de protección a los sistemas y los recursos. Esta protección extra permite que se prevengan numerosos ciberataques que perjudican especialmente a los usuarios.

Métodos de autorización más utilizados

1. Autorización HTTP

- **Descripción**: Este método se basa en el uso de encabezados HTTP para enviar credenciales de usuario (nombre de usuario y contraseña) al servidor.
- **Funcionamiento**: Si las credenciales son incorrectas, el servidor responde con un código de estado 401 "Unauthorized". El cliente puede entonces volver a enviar una solicitud con un encabezado Authorization que incluya las credenciales.
- **Ventajas**: Es un método sencillo y directo, ideal para servicios que no requieren sesiones o cookies.

2. Autorización API

- **Descripción**: Al registrarse en un servicio, se genera una clave API para el usuario, que se utiliza para acceder a recursos específicos.
- **Funcionamiento**: La combinación de clave API y un token (que actúa como identificador) permite al cliente autenticar sus solicitudes al servidor. Esta combinación se utiliza en cada interacción para definir el acceso y la respuesta del servidor.
- **Ventajas**: Proporciona una manera estandarizada de autenticar y autorizar solicitudes en entornos de API, facilitando la integración entre sistemas.

3. OAuth 2.0

- **Descripción**: Un protocolo que permite la autorización segura de aplicaciones de terceros para acceder a los recursos de un usuario sin compartir sus credenciales.
- **Funcionamiento**: El usuario otorga permiso a una aplicación para realizar acciones en su nombre, usando un token de acceso. Este token se utiliza para acceder a recursos protegidos en el servidor.
- **Ventajas**: Proporciona una forma segura y flexible de manejar la autorización, permitiendo acceso granular a recursos específicos.

4. Autorización JWT (JSON Web Token)

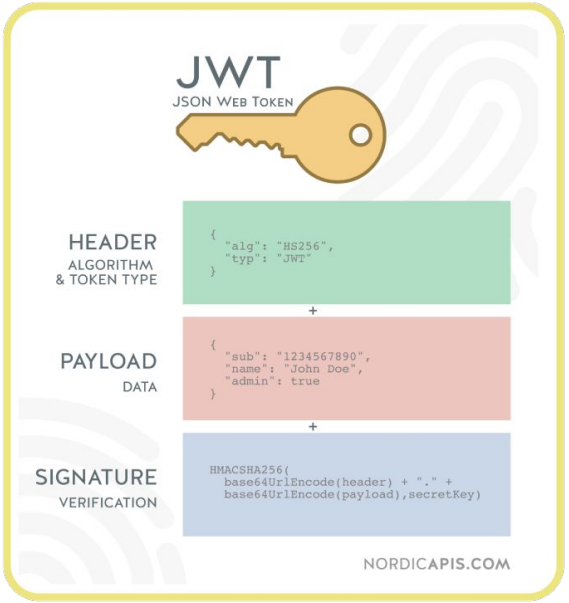
- **Descripción**: Un estándar abierto que permite la transmisión segura de información entre diferentes partes como un objeto JSON, que puede ser verificado y confiado.
- **Funcionamiento**: Se utiliza un par de claves (privada y pública) para firmar y verificar el token. El JWT contiene información sobre el usuario y sus permisos, y se envía en cada solicitud.
- **Ventajas**: Permite la autenticación y autorización de manera descentralizada y escalable, y es ampliamente utilizado en aplicaciones modernas.

Json Web Token (JWT)

JSON Web Token (JWT) es un estándar abierto que permite la creación de tokens de seguridad para autenticar y autorizar a los usuarios en aplicaciones web y servicios. JWT puede ser utilizado en dos formas: como JSON Web Signature (JWS) o como JSON Web Encryption (JWE).

> JSON Web Signature (JWS) es una forma de firmar digitalmente un token JWT para garantizar su integridad y autenticidad. La firma se basa en un algoritmo de hash y una clave secreta compartida. El receptor del token puede verificar la firma utilizando la clave secreta y el algoritmo de hash especificado en el encabezado del token.
>
> JSON Web Encryption (JWE) es una forma de cifrar el contenido de un token JWT para proteger la privacidad de los datos. El contenido cifrado se puede descifrar solo con una clave secreta específica. JWE utiliza algoritmos de cifrado para proteger la información.

Podemos utilizar dos tipos de algoritmos para codificar nuestros tokens, los simétricos nos permiten encriptar y desencriptar los tokens utilizando una única llave privada, o podemos utilizar algoritmos asimétricos que utilizan una llave privada y una pública para tener mayor seguridad y evitar problemas si alguna de las llaves es interceptada.



¿Qué es JSON Web Token (JWT)?

JSON Web Token (JWT) es un estándar abierto que permite la creación de tokens compactos y seguros para la autenticación y autorización en aplicaciones web y servicios. Los JWT pueden ser utilizados para transmitir información de forma segura entre partes como un objeto JSON.

Tipos de JWT

1. **JSON Web Signature (JWS)**:
   - **Descripción**: Proporciona integridad y autenticidad al token mediante la firma digital.
   - **Funcionamiento**: Utiliza un algoritmo de hash y una clave secreta compartida para firmar el token. El receptor puede verificar la firma utilizando la misma clave.
2. **JSON Web Encryption (JWE)**:
   - **Descripción**: Cifra el contenido del token para proteger la privacidad de los datos.
   - **Funcionamiento**: Utiliza algoritmos de cifrado para asegurar que solo las partes autorizadas puedan descifrar y acceder al contenido del token.

Algoritmos de Codificación

- **Simétricos**: Usan una única clave secreta para encriptar y desencriptar el token. Ejemplo: HMAC.
- **Asimétricos**: Utilizan un par de claves (una privada y una pública). La clave privada firma el token, y la clave pública se utiliza para verificarlo.

Estructura de un JWT

- La primera parte es el header y contiene el algoritmo y el tipo de token que utilizamos para firmar.

- La segunda parte es el payload, contiene información como la identificación del usuario, fechas de creación y expiración del token, entre otras (debemos tener cuidado de no transmitir información sensible ya que puede ser decodificada por alguna otra aplicación).

- Por último, la signature es la tercera parte del token y se genera codificando los anteriores campos más una firma secreta. Gracias a esta parte del token podemos verificar su autenticidad e invalidar el token si alguno de los campos cambia.

Las partes de un JWT se codifican como un objeto JSON que está firmado digitalmente utilizando JSON Web Signature (JWS). Los JWT se dividen en 3 cadenas de texto separadas por puntos. Cada una de las partes está codificada en base64Url.

Los tokens JWT se componen de tres secciones: un algoritmo de firmado, el payload (información) y la firma (llave que permite al receptor verificar que el mensaje no ha sido modificado durante la transmisión)

Base64Url

Es una variante del esquema de codificación Base64, utilizada principalmente en el contexto de la transmisión de datos en aplicaciones web y sistemas que requieren una representación de datos segura y eficiente, como JSON Web Tokens (JWT).
¿Como Funciona? El algoritmo toma cualquier dato, lo pasa a binario, y va tomando porciones de 6 bits (ya que 2^6=64), que en decimal nos indica el índice (empezando en 0) del carácter equivalente en el alfabeto elegido.

Características de Base64Url

1. **Codificación Base64**:
   - Base64 es un método de codificación que convierte datos binarios en una cadena de texto utilizando 64 caracteres ASCII. Esto es útil para transmitir datos en formatos que solo aceptan texto, como JSON o URLs.
2. **Diferencias con Base64**:
   - **Caracteres en Base64Url**:
     - Base64 utiliza `+` y `/` como caracteres adicionales, mientras que Base64Url sustituye `+` por `-` y `/` por `_`.
   - **Eliminación de relleno**:
     - Base64 puede añadir uno o dos caracteres de relleno (`=`) al final para completar el bloque de 4 caracteres. Base64Url elimina estos caracteres de relleno, lo que simplifica su uso en URLs.
3. **Usos Comunes**:
   - **JWT**: Base64Url se usa para codificar las partes del JWT (header, payload y signature) para que sean seguras y compatibles con las URLs.
   - **APIs REST**: Se utiliza para codificar datos en solicitudes y respuestas sin problemas de compatibilidad.

Estructura de un JWT

Un JWT está compuesto por tres partes, codificadas en Base64Url y separadas por puntos (`.`):

1. **Header (Encabezado)**:
   - Contiene el tipo de token (JWT) y el algoritmo de firma utilizado (por ejemplo, HMAC SHA256).
2. **Payload (Carga útil)**:
   - Contiene la información que se desea transmitir. Puede incluir datos como:
     - `sub` (subject): Identificación del usuario.
     - `iat` (issued at): Fecha de creación del token.
     - `exp` (expiration): Fecha de expiración del token.
   - **Nota**: Debe evitarse incluir información sensible, ya que el payload puede ser decodificado fácilmente.
3. **Signature (Firma)**:
   - Se genera combinando el encabezado y la carga útil, y luego firmándolos con la clave secreta (en el caso de JWS) o cifrando (en el caso de JWE).
   - La firma permite al receptor verificar que el token no ha sido modificado durante la transmisión.

Ejemplo de Estructura de un JWT

`eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c`

- **Header**: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9`
- **Payload**: `eyJzdWIiOiIxMjM0NTY3ODkwIiwiaWF0IjoxNTE2MjM5MDIyfQ`
- **Signature**: `SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c`

Ventajas de usar JWT

- **Compacto**: Se puede transmitir fácilmente a través de URL, parámetros POST y cabeceras HTTP.
- **Seguro**: Permite la verificación de la autenticidad y la integridad del mensaje.
- **Escalable**: Ideal para aplicaciones distribuidas y microservicios, ya que el estado no necesita ser mantenido en el servidor.

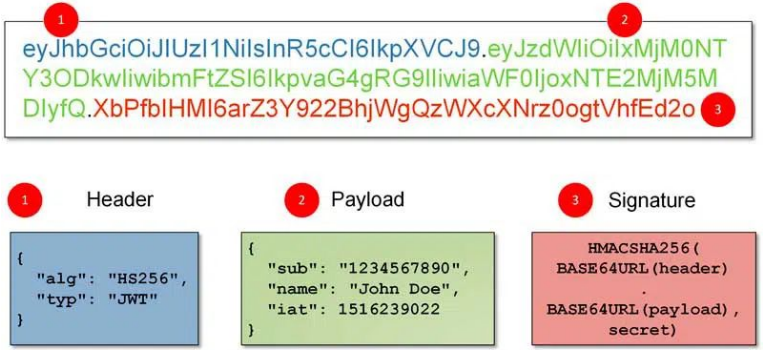
Token JWT en Node

jsonwebtoken es un estándar abierto que se utiliza para compartir información entre dos partes: un cliente y un servidor. Usaremos dos funciones de JWT . La primera función es sign para crear un nuevo token y la segunda función es verify para verificar el token.

Instalamos la librería npm install jsonwebtokenjwt.sign() es un método que recibe dos parámetros:

> El primer parámetro es el payload, que es un objeto con la información de autenticación y/o autorización que queremos incluir en el token, en este caso contiene una propiedad username con el valor 'admin'.
>
> El segundo parámetro es la clave secreta que se utilizará para firmar el token.

Y estas 3 partes, cuando se combinan, forman un JW-Token que tiene un aspecto similar al siguiente:



RSA 256

RSA (Rivest-Shamir-Adleman) es un algoritmo de cifrado de clave pública desarrollado en 1977. RSA 256 es una variante de RSA que utiliza una clave de 256 bits (32 bytes) para cifrar y descifrar los datos. RSA 256 es considerado seguro y es utilizado en muchos sistemas de seguridad, incluyendo la seguridad de la información en internet.

Genera dos claves: una clave pública y una clave privada. La clave pública se utiliza para cifrar los datos, mientras que la clave privada se utiliza para descifrarlos. Los datos cifrados con la clave pública sólo pueden ser descifrados con la clave privada correspondiente.

¿Cómo firmar un token JWT con RSA 256?

La forma de verificar tokens JWT firmados con un algoritmo asimétrico como RSA, es usando la llave pública. En nuestro caso será usando la llave que está dentro del archivo jwtRS256.key.pub

¿Podemos ver el contenido de un token JWT sin tener la llave pública?

Si. La misión de los tokens JWS es garantizar que la información que es enviada desde el emisor hasta el receptor no sea modificada durante la transmisión. Esto no quiere decir que esta información vaya cifrada, por el contrario, quien obtenga un token JWT puede ver todo su contenido, lo cual no debería ser un problema si se está utilizando esta estrategia.

Para ver el contenido de un token JWT, podemos copiarlo e ingresar al sitio: https://jwt.io/

Expiración y Audiencia

El atributo expiresIn está dado en segundos, por tanto, en la anterior implementación el token durará una hora. También es posible indicar este valor en otras unidades, como por ejemplo “1h” o “150ms” .

Audiencia:Imaginemos un escenario donde creamos nuestro propio servicio de manejo de identidad. El objetivo es que diferentes aplicaciones utilicen ese servicio para manejar la autenticación basada en tokens JWT . Dado que emitimos los tokens con la misma llave privada RSA, si alguien genera un token en la aplicación A, este también será un token válido para la aplicación B, lo cual estaría dejando expuestas todas las aplicaciones que usan el servicio a una vulnerabilidad grave.

La audiencia de los tokens busca mitigar esta vulnerabilidad, haciendo que cuando se emita un token, se indique para quién fue generado. Por tanto, si se emite un token para la aplicación A no podrá ser utilizado para consumir recursos de la aplicación B.

En muchas ocasiones en la audiencia se define el clientId de la aplicación para la cual se está generando el token (pero en realidad puede ser cualquier String).

Módulo Bcrypt

Utilizamos el módulo bcrypt para cifrar la contraseña del usuario antes de guardarla en la base de datos. La contraseña enviada en la solicitud no se almacena en la base de datos. Almacenamos el hash de la contraseña que se genera con la función bcrypt.hash .

Las saltRounds son un parámetro utilizado en la función de hash de contraseñas bcrypt . Es el número de veces que se ejecuta el algoritmo de hash. A medida que aumenta el número de rondas de sal, se hace más difícil para un atacante descifrar las contraseñas almacenadas en la base de datos. Sin embargo, también aumenta el tiempo que se tarda en generar el hash, por lo que se debe buscar un equilibrio adecuado.

Usuarios únicos

Por defecto nos va a instalar la versión más actual, en este caso 8.7.2 de mongoose, El error que puede aparecer al intentar instalar mongoose-unique-validator se debe a un conflicto de versiones entre las dependencias de mongoose y mongoose-unique-validator.

Conflicto de Dependencias:

- mongoose-unique-validator (versión 4.0.1) tiene una dependencia peer que requiere mongoose en una versión que sea compatible con ^7.0.0.
- Al tener instalada la versión 8.7.2 de mongoose, no es compatible con la dependencia especificada por mongoose-unique-validator por el momento.

Soluciones: Instalar una versión anterior de Mongoose:

Si no necesitas las características de la versión 8 de mongoose, podrías bajar la versión a 7.x.x. Puedes hacer esto con: **npm install mongoose@^7.0.0**

Luego intenta nuevamente instalar mongoose-unique-validator: **npm install mongoose-unique-validator**

PasswordHash

Generalmente se refiere a un valor que representa una contraseña en forma hash, utilizado para almacenar contraseñas de manera segura en bases de datos. En lugar de guardar la contraseña en texto claro, se guarda un valor hash, lo que dificulta que un atacante obtenga la contraseña original incluso si logra acceder a la base de datos.

¿Cómo Funciona el Hashing de Contraseñas?

1. **Hashing**: Cuando un usuario crea una cuenta o cambia su contraseña, se aplica una función de hash a la contraseña. Esta función toma la contraseña original y produce una cadena de longitud fija, que es el hash.
2. **Salting**: Para aumentar la seguridad, se puede agregar un "sal" (una cadena aleatoria) a la contraseña antes de aplicar la función de hash. Esto significa que incluso si dos usuarios tienen la misma contraseña, el hash será diferente debido a los diferentes salts.
3. **Almacenamiento**: El hash (y el salt, si se utiliza) se almacena en la base de datos en lugar de la contraseña en texto claro.
4. **Verificación**: Cuando un usuario intenta iniciar sesión, se toma la contraseña proporcionada, se aplica la misma función de hash (y el mismo salt) y se compara el resultado con el hash almacenado. Si coinciden, la contraseña es correcta.

Ventajas de Usar Hashing para Contraseñas

- **Seguridad**: Almacenar hashes en lugar de contraseñas en texto claro protege las credenciales de los usuarios en caso de una brecha de seguridad.
- **Imposibilidad de revertir**: Las funciones de hash son unidireccionales, lo que significa que no se puede recuperar la contraseña original a partir del hash.

Bearer

Bearer es un tipo de esquema de autenticación utilizado en la autenticación de recursos HTTP. En este esquema, un token es incluido en el encabezado de la solicitud HTTP como un valor de la autorización. El servidor recibe el token de portador y lo utiliza para validar la solicitud y determinar el usuario autenticado.

Sintaxis de uso del token en una solicitud HTTP para incluir un token en una solicitud HTTP es "Authorization: Bearer <token>".

En la práctica, esto significa que si el token es, por ejemplo, la cadena eyJhbGciOiJIUzI1NiIsInR5c2VybmFtZSI6Im1sdXVra2FpIiwiaW , el encabezado de autorización tendrá el valor:

Buenas prácticas al utilizar JWT

- No transmitir información sensible: No se debe transmitir información sensible a través de JWT, ya que estos son completamente decodificables. Debemos tratar toda la información transmitida como si fuera enviada en texto plano.
- Mantener los tokens pequeños: Los JWT no son un medio de transmisión de datos sino una forma de verificar la autenticación. Para obtener información de usuarios, se deben crear endpoints en la API que estén disponibles solo si se envía un token válido.
- Configurar tiempos de vida cortos: Es recomendable configurar tiempos de vida cortos para los tokens, ya que cuanto mayor sea el tiempo de vida de un token, mayor es el riesgo de sufrir un ataque.
- Crear JWT opacos: No se deben decodificar los tokens desde el cliente o frontend, ya que el código es público y existe el riesgo de que alguien acceda a las llaves privadas.