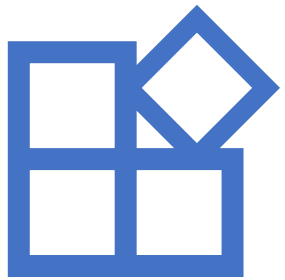


Autenticación con JWT

Autenticación

- La **autenticación** es una de las piezas claves, ya que la seguridad depende en gran medida de este punto.
- La autenticación con tokens supuso un gran
- El **refresh token** llegó para complementarla y hacerla usable.

Sistemas de autenticación según verificación de usuario

Basados en algo conocido (password)

Basados en algo poseído (tarjeta de identidad, usb, token)

Basados en características físicas (voz, huellas, ojos)

Autenticación basada en tokens

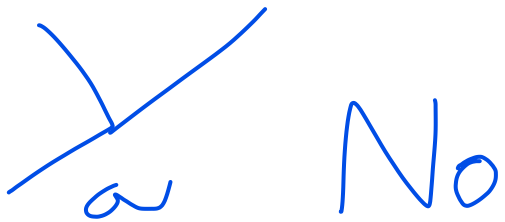
Y en APIs

- Introducidos en aplicaciones web por la **autenticación y autorización moderna**.
- Su uso se extendió gracias al protocolo OAuth (posteriormente OAuth2)
- Oauth esta centrado en la autorización, y no en la autenticación

Clasificación autenticación con tokens

Autenticación tradicional o autenticación en servidor

Autenticación sin estado basada en tokens



Autenticación tradicional o autenticación en servidor

- Cuando un usuario se loguea, el servidor le devuelve un token que típicamente es almacenado en una cookie.
- El servidor guarda la información de la **sesión**, bien en memoria o en base de datos (Redis, MongoDB …).
- Cada vez que el usuario hace una petición con ese token, el servidor busca la información para saber qué usuario está intentando acceder y si es válida, ejecuta el método solicitado.

Problemas de la autenticación tradicional o autenticación en servidor

SOBRECARGA PROVOCADA POR TODA LA INFORMACIÓN DE LOS USUARIOS AUTENTICADOS.

ESCALABILIDAD, YA QUE SI HAY VARIAS INSTANCIAS DEL SERVIDOR LEVANTADAS, TENDRÍAN QUE COMPARTIR DE ALGÚN MODO LA INFORMACIÓN DE LA SESIÓN PARA NO HACERLE LOGARSE DE NUEVO.

VULNERABILIDADES DEBIDAS A ESTA ARQUITECTURA (CORS, CSRF)

Esto sí

Autenticación sin estado basada en tokens

- **Autenticación sin estado** (*stateless*). Surge para solucionar estos inconvenientes.
- El servidor no va a almacenar ninguna información, ni tampoco la sesión.
- Cuando el usuario se autentica con sus credenciales o cualquier otro método, en la respuesta recibe un token (**access token**).
- Todas las peticiones que se hagan al API llevarán este token en una cabecera HTTP
- El servidor podrá identificar qué usuario hace la petición sin necesidad de buscar en base de datos ni en ningún otro sistema de almacenamiento.

Ventajas de la autenticación sin estado basada en tokens

- **Escalabilidad.** Es el propio cliente el que almacena su información de autenticación, y no el servidor.
- **Sincronizaciones**. Las peticiones pueden llegar a cualquier instancia del servidor y podrá ser atendida sin necesidad de sincronizaciones.
- **Diferentes plataformas** podrán usar el mismo API
- **Incrementa la seguridad**, evitando vulnerabilidades CSRF, al no existir sesiones.
- **Expiración**. Si añadimos expiración al token la seguridad será aún mayor.

Obligatoria para los tokens

JWT (JSON Web Token)

- JSON Web Token (JWT) es un estandar abierto basado en JSON para crear tokens de acceso que permiten el uso de recursos de una aplicación o API.
- Este token llevará incorporada la información del usuario que necesita el servidor para identificarlo, así como información adicional que pueda serle útil (roles, permisos, etc.).

JWT (JSON Web Token)

- **JWT**: formato interno a usar por la información almacenada en el token.
- **JWS y JWE**. Útil si se usa junto a JSON Web Signature (JWS) y JSON Web Encryption (JWE).
  - La combinación de **JWT junto con JWS y JWE** nos permite no sólo autenticar al usuario, sino enviar la información **encriptada** para que sólo el servidor pueda extraerla, así como validar el contenido y asegurarse que no ha habido suplantaciones o modificaciones.

Refresh Token

- **Tiempo de validez.** Una vez pasado el tiempo de validez, el servidor no permitirá más el acceso a recursos con dicho token.
- **Refresh Token**. El usuario tendrá que conseguir un nuevo access token volviéndose a autenticar o con algún método adicional: **refresh token**.

Composición del token

- Un token JWT está formado por 3 partes separadas por un . siendo cada una de ellas:
  - **cabecera (header):** con el tipo (JWT) y el tipo de codificación
  - **Cuerpo (payload):** Es donde se encontrará la información del usuario que permitirá al servidor discernir si puede o no acceder al recurso solicitado
  - **Firma de verificación (signature):** Se aplicará la función de firmado a los otros dos campos del token para obtener el campo de verificación

para verificar que el token es correcto

Cabecera token JWT

- Contiene el tipo (JWT) y el tipo de codificación

```
{
   "alg": "HS256",
   "typ": "JWT"
}
```

Cuerpo token JWT

- Es donde se encontrará la información del usuario que permitirá al servidor discernir si puede o no acceder al recurso solicitado

```
{
  username: 'john_doe',
  email: 'john_doe@server.com',
  name: 'John Doe',
  role: 'user',
  exp: 1478773621
}
```

Tipos de token

- **Access token:** Token de acceso. Contiene toda la información que necesita el servidor para saber si el usuario / dispositivo puede acceder al recurso que está solicitando o no.
- **Refresh token:** Token de refresco. Utilizado para generar un nuevo access token

Access Token

- Lleva contenida toda la información que necesita el servidor para saber si el usuario / dispositivo puede acceder al recurso que está solicitando o no.
- Suelen ser tokens caducos con un periodo de validez corto.

Refresh Token (I)

- El refresh token es usado para generar un nuevo access token.
- Si el access token tiene fecha de expiración, una vez que caduca, el usuario tendría que autenticarse de nuevo para obtener un access token.
- Con el refresh token, este paso se puede saltar y con una petición al API obtener un nuevo access token que permita al usuario seguir accediendo a los recursos de la aplicación.

Refresh Token (II)

- También puede ser necesario generar un nuevo access token cuando se quiere acceder a un recurso que no se ha accedido con anterioridad
- El refresh token requiere una seguridad mayor a la hora de ser almacenado que el access token, ya que si fuera sustraido por terceras partes, podrían utilizarlo para obtener access tokens y acceder a los recursos protegidos de la aplicación.
- Para poder cortar un escenario como este, debe implementarse en el servidor algún sistema que permita **invalidar un refresh token**, además de establecer un **tiempo de vida** que obviamente debe ser más largo que el de los access tokens.