



ALMACENAMIENTO SEGURO DE JWT



¿DONDE ALMACENAR LOS TOKENS EN EL FONRT

Al elegir un lugar para almacenar los tokens JWT en el frontend, es crucial considerar la seguridad contra ataques como XSS (Cross-Site Scripting) y CSRF (Cross-Site Request Forgery). A continuación, se presentan las opciones más comunes con sus ventajas y desventajas:





Ventajas:

- Fácil de usar y acceder desde cualquier parte del frontend.
- Persiste después de que el usuario recarga la página o cierra el navegador.

Desventajas:

- Altamente vulnerable a ataques XSS: Si un atacante inyecta código malicioso en la página, podría extraer el token.
- No se envía automáticamente en las solicitudes HTTP, lo que requiere incluirlo manualmente en los headers.

LOCALSTORAGE

Almacena datos de forma persistente en el navegador, incluso después de que se cierre la pestaña o el navegador.

Cuando usarlo?

- No recomendado para almacenar JWT. Si es necesario, solo para datos no sensibles.



SESSIONSTORAGE

Similar a localStorage, pero los datos se eliminan automáticamente cuando se cierra la pestaña o el navegador.

Cuando usarlo?

Si necesitas que el token solo dure la sesión del usuario y no es crítico protegerlo contra XSS.

Ventajas:

- No persiste después de cerrar la pestaña o navegador (mejor que localStorage).
- Reduce el riesgo en caso de robo del token en una sola sesión.

Desventajas:

- Sigue siendo vulnerable a ataques XSS.
- No se comparte entre pestañas o ventanas abiertas del navegador.



HTTP-only Cookies

Son cookies que solo pueden ser accedidas por el servidor, no por JavaScript en el navegador

Ventajas

- Protección contra XSS: JavaScript no puede acceder al token.
- Se envía automáticamente en cada solicitud HTTP, lo que facilita la autenticación.

Desventajas

- Requiere configuración en el backend.
- Puede ser afectado por restricciones de cookies en navegadores modernos.



Cuando usarlo?

Para autenticación segura y persistente sin exponer el token al frontend.

BUENA PRÁCTICAS RECOMENDADAS

- Almacenar Access Tokens en memoria (evita ataques XSS)

¿Cómo almacenar el Access Token en memoria?

Almacenar el Access Token en memoria significa que el token solo vive dentro del estado de la aplicación mientras la página esté abierta. No se guarda en localStorage ni en sessionStorage, lo que evita ataques XSS (Cross-Site Scripting).

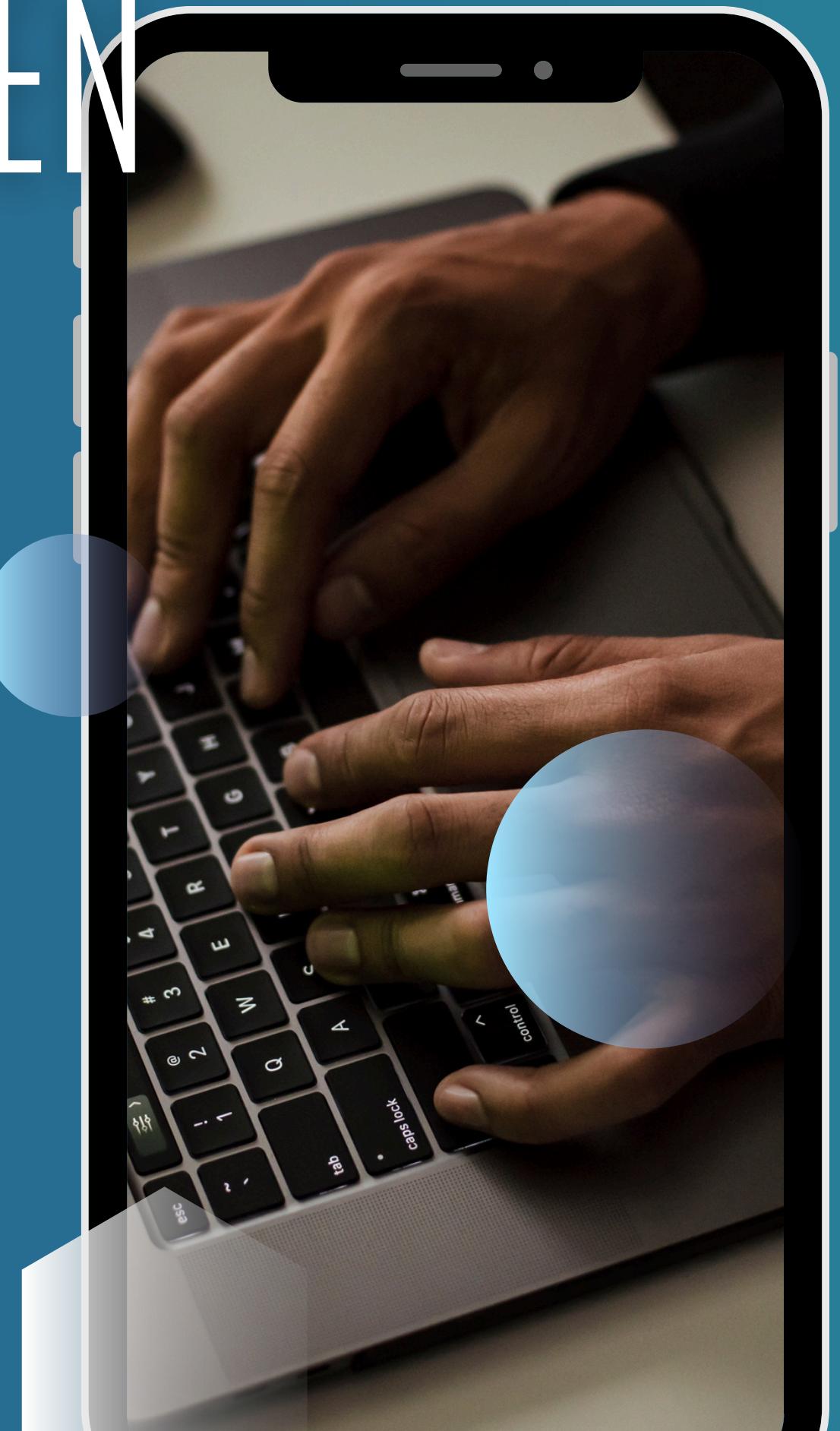
Ver ejemplo en Repositorio: [proyectos-> frontend_access_token](#)



CONCEPTO DE REFRESH TOKEN

Un Refresh Token es un tipo de token utilizado en sistemas de autenticación para obtener nuevos Access Tokens cuando el Access Token original ha expirado.

A diferencia de un Access Token, que tiene una vida útil relativamente corta (por razones de seguridad), el Refresh Token generalmente tiene una vida útil más larga y puede ser utilizado para obtener nuevos Access Tokens sin necesidad de que el usuario vuelva a autenticarse.



COMO FUNCIONA EL REFRESH TOKEN

Ver repositorio: Proyectos → Frontend_refresh_token

Inicio de sesión:

El usuario inicia sesión en la aplicación y, al hacerlo, el servidor genera un Access Token (que generalmente tiene una vida corta, como 15 minutos a 1 hora) y un Refresh Token (que puede durar días, semanas o incluso meses)

Espiración del Access Token:

Cuando el Access Token expira (por ejemplo, después de 1 hora), el cliente ya no puede acceder a los recursos protegidos con ese token.

Explicación del flujo:

- Cuando el usuario inicia sesión, el backend envía el Refresh Token en una cookie HttpOnly.
- El Access Token se almacena en memoria para protegerlo de ataques XSS.
- Cada vez que el frontend hace una solicitud, usa el Access Token en los encabezados.
- Si el Access Token expira (error 401), el frontend llama a refreshToken():
- Envía el Refresh Token en una cookie HttpOnly (el backend valida y responde con un nuevo Access Token).
- El frontend almacena el nuevo Access Token y reintenta la solicitud original.
- Si la renovación falla (por ejemplo, Refresh Token expirado), el usuario es redirigido a la página de inicio de sesión.