JWT

JWT (JSON Web Token) es un estándar abierto (RFC 7519) para la creación de tokens de acceso que permiten la transmisión segura de información entre partes como un objeto JSON firmado digitalmente.

Características principales:

- **Autenticación**: Se usa comúnmente para autenticación en aplicaciones web y APIs.
- **Compacto**: Su estructura ligera lo hace ideal para ser transmitido en encabezados HTTP o almacenado en cookies.
- **Seguro**: Puede ser firmado usando HMAC o cifrado con RSA/ECDSA para garantizar la integridad y autenticidad.

Estructura de un JWT:

Un token JWT consta de tres partes separadas por puntos (`.`):

1. **Header (Encabezado)**: Contiene el tipo de token (`JWT`) y el algoritmo de firma (por ejemplo, `HS256` para HMAC con SHA-256).
2. **Payload (Cuerpo o Carga útil)**: Contiene los claims (información sobre el usuario o permisos). Puede incluir claims estándar como `sub` (sujeto), `exp` (expiración), `iat` (emitido en), y claims personalizados.
3. **Signature (Firma)**: Se genera firmando el header y el payload con una clave secreta o un par de claves pública/privada.

Ejemplo de un JWT:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjEsIm5hbWUiOiJKb2huIERvZSIsImV4cCI6MTY5MDUyNTAwMH0.pHjP1V5K8kzzKPn0shVW6hZGQvHtrI6HtJ0kZuXtXVk
```

Uso en autenticación:

1. **El usuario inicia sesión** y el servidor genera un JWT con información sobre el usuario.
2. **El cliente almacena el JWT** (en `localStorage`, `sessionStorage` o una cookie segura).
3. **El cliente envía el JWT en cada solicitud** (normalmente en el encabezado `Authorization: Bearer <token>`).
4. **El servidor verifica el JWT** y, si es válido, permite el acceso a los recursos protegidos.

Ventajas:

✅ Stateless (sin necesidad de almacenamiento en el servidor)
✅ Seguridad con firma digital
✅ Rápido y eficiente en la autenticación

Desventajas:

❌ Si se compromete el token, cualquier persona con acceso puede usarlo hasta que expire.
❌ No se puede invalidar fácilmente (se recomienda manejar una lista de revocación o usar tokens de corta duración con refresh tokens).

Una API puede obtener un **JWT (JSON Web Token)** de varias maneras dependiendo del método de autenticación que se implemente. A continuación, te explico las formas más comunes:

A través del Header HTTP (`Authorization` Header)

Este es el método más utilizado en APIs REST y GraphQL.

- **Cómo se envía:**
  En el cliente (frontend o consumidor de la API), se envía el JWT en cada solicitud dentro del **encabezado** `Authorization`.

Ejemplo de cabecera HTTP:

```
Authorization: Bearer <jwt_token>
```

-

Cómo la API lo obtiene en Express.js (Node.js):

```
const token = req.headers.authorization?.split(" ")[1];
```

- **Ventajas:**

  ✅ Es seguro si se usa HTTPS.
  ✅ No se expone en cookies ni `localStorage`.
  ✅ Ideal para APIs sin estado (stateless).

- **Desventajas:**

  ❌ Requiere que el cliente siempre envíe el token en cada solicitud.

A través de Cookies (Autenticación basada en sesiones)

Se puede almacenar el JWT en una **cookie HTTP-only** y la API lo obtiene automáticamente en cada petición.

Cómo se envía:
Cuando el usuario inicia sesión, el servidor genera el JWT y lo almacena en una **cookie segura**:
ts
CopiarEditar

```
res.cookie("token", jwtToken, {
  httpOnly: true,
  secure: true, // Solo en HTTPS
  sameSite: "Strict",
});
```

-

Cómo la API lo obtiene en Express.js (Node.js):
ts
CopiarEditar

```
const token = req.cookies.token;
```

- **Ventajas:**

  ✅ Protege contra ataques XSS.
  ✅ No necesita que el frontend agregue manualmente el token a cada solicitud.

- **Desventajas:**

  ❌ Vulnerable a ataques CSRF (se recomienda usar `SameSite=Strict`).
  ❌ No funciona bien en APIs móviles o de terceros.

Tipos de Información que se Puede Guardar en un JWT

El **Payload** del JWT es un JSON que generalmente contiene **claims** (declaraciones sobre el usuario o sesión). Estos claims pueden ser:

Claims Estándar (Recomendados por el RFC 7519)

Son campos predefinidos que facilitan la interoperabilidad con otros sistemas:

Claim	Descripción
`iss` (Issuer)	Identifica quién emitió el token (ej. `api.example.com`).
`sub` (Subject)	Identifica al usuario o entidad (ej. `user_id`).
`aud` (Audience)	Define para quién está destinado el token (ej. `frontend.example.com`).
`exp` (Expiration)	Fecha de expiración en **timestamp Unix**.
`iat` (Issued At)	Cuándo fue emitido el token en **timestamp Unix**.
`nbf` (Not Before)	Fecha antes de la cual el token no es válido.
`jti` (JWT ID)	ID único del token (para evitar reutilización).

Ejemplo de Payload con claims estándar:

json
CopiarEditar

```
{
  "iss": "api.example.com",
  "sub": "1234567890",
  "aud": "frontend.example.com",
  "exp": 1690900800,
  "iat": 1690897200
}
```

Claims Personalizados (Información Específica de la Aplicación)

Además de los claims estándar, puedes agregar información personalizada. Sin embargo, **no almacenes datos sensibles sin encriptarlos primero**.

Claim Personalizado	Descripción
`userId`	Identificador único del usuario en la base de datos.
`role`	Rol del usuario (ej. `admin`, `user`, `moderator`).
`permissions`	Lista de permisos específicos del usuario.
`email`	Correo electrónico del usuario.
`username`	Nombre de usuario.
`organizationId`	ID de la empresa u organización a la que pertenece el usuario.

Ejemplo de Payload con claims personalizados:

json
CopiarEditar

```
{
  "sub": "1234567890",
  "userId": "98765",
  "role": "admin",
  "permissions": ["create", "read", "update", "delete"],
  "email": "user@example.com",
  "organizationId": "55"
}
```

❌ Información que NO Deberías Guardar

Guardar ciertos datos en un JWT puede comprometer la seguridad de tu aplicación:

1. ⚠️ **Datos Sensibles**
   - Contraseñas (ni siquiera encriptadas).
   - Tarjetas de crédito o información bancaria.
   - Tokens de sesión u otras credenciales.
2. 📏 **Información Grande**
   - Evita guardar datos muy grandes (como imágenes o listas de compras).
   - JWT debe ser compacto para ser eficiente.

3. 🕵️ **Información que Pueda Cambiar con Frecuencia**
    - Como el balance de una cuenta, ya que JWT no puede ser editado después de emitirse.
    - En estos casos, mejor usa una base de datos y solo almacena el `userId`.

✅ Buenas Prácticas

✔ Usa **tokens de corta duración** y refresh tokens para mantener la seguridad.
✔ Firma siempre el JWT con una clave segura.
✔ **No confíes en los datos dentro del JWT sin validarlo primero**.
✔ Usa **cookies HTTP-only** en vez de `localStorage` si el JWT es para autenticación en navegadores.
✔ Limita el uso de claims personalizados solo a la información necesaria.