JWT, o **JSON Web Token**, es un estándar para representar de manera segura información entre dos partes en forma de un token JSON. Este token es compacto, seguro y firmado digitalmente, lo que permite verificar tanto la identidad de los usuarios como la integridad de los datos transferidos.

¿Para qué sirve JWT?

JWT se usa principalmente para **autenticación y autorización** en aplicaciones web. Su propósito es asegurar que, una vez que un usuario se autentique (por ejemplo, iniciando sesión), el sistema pueda reconocerlo en futuras solicitudes sin necesidad de almacenar su sesión en el servidor.

¿Cómo funciona JWT?

1. **Generación del token**: Cuando un usuario se autentica con éxito, el servidor crea un JWT que contiene información relevante sobre el usuario (como su ID o roles) y lo envía al cliente. Esta información se almacena en el **payload** del token.
2. **Estructura del token**: Un JWT consta de tres partes, separadas por puntos:
    - **Header (encabezado)**: Describe el tipo de token y el algoritmo de cifrado (generalmente HMAC SHA256 o RSA).
    - **Payload (carga)**: Contiene los datos (o **claims**) que queremos transferir, como el ID de usuario, nombre y permisos.
    - **Signature (firma)**: Es una combinación del header y payload codificados y firmados con una clave secreta del servidor. Esta firma asegura que el token no haya sido modificado.

Un JWT típico se vería así:

xxxxx.yyyyy.zzzzz

1. **Verificación del token**: En cada solicitud a una ruta protegida, el cliente envía el token JWT (generalmente en el encabezado Authorization como Bearer <token>). El servidor verifica la firma del token utilizando la clave secreta para comprobar su autenticidad. Si el token es válido, se permite el acceso a los recursos protegidos.
2. **Expiración y seguridad**: Los tokens JWT suelen tener un tiempo de expiración (por ejemplo, 1 hora). Esto añade una capa de seguridad, ya que el token caduca y el usuario debe autenticarse nuevamente para obtener un nuevo token.

Ventajas de usar JWT

- **Escalable y sin estado**: No requiere almacenar sesiones en el servidor, lo que facilita la escalabilidad en aplicaciones distribuidas.
- **Seguridad**: La firma digital garantiza que los datos no se hayan modificado, y los datos sensibles pueden mantenerse encriptados.
- **Facilidad de uso en APIs**: Ideal para sistemas basados en APIs RESTful, donde el token se pasa en cada solicitud para verificar la autenticación.

Casos de uso de JWT

- **Autenticación de usuarios**: Tras un inicio de sesión exitoso, el servidor devuelve un JWT al cliente, que lo usa en cada solicitud.
- **Autorización de acceso**: Basado en roles o permisos en el payload del JWT, el servidor puede controlar el acceso a recursos específicos.
- **Intercambio de información seguro**: JWT también se utiliza para transferir información entre servicios de manera segura en microservicios o sistemas distribuidos.

En resumen, JWT es una solución eficiente para manejar autenticación y autorización en aplicaciones modernas, particularmente en entornos de APIs y microservicios.

COMO UTILIZAR EL JWT CON EXPRESS Y NODE JS.

Para utilizar JWT (JSON Web Token) en una aplicación Node.js con Express, puedes seguir estos pasos para manejar la autenticación de usuarios de manera segura:

Instala las dependencias necesarias
Primero, asegúrate de tener las siguientes librerías instaladas:

npm install express jsonwebtoken bcryptjs dotenv

- **jsonwebtoken**: para crear y verificar tokens JWT.
- **bcryptjs**: para encriptar las contraseñas de los usuarios.
- **dotenv**: para manejar variables de entorno como la clave secreta.

Configura las variables de entorno
Crea un archivo .env en la raíz del proyecto para almacenar tu clave secreta. Agrega una variable de entorno como:

JWT_SECRET=your_secret_key

Configura tu servidor en Express
En tu archivo principal de servidor, carga las variables de entorno y configura Express:

```
require('dotenv').config();
const express = require('express');
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs');

const app = express();
app.use(express.json()); // Para parsear JSON

const PORT = process.env.PORT || 3000;
```

Crea el sistema de autenticación
Registro de usuario

Define una ruta para el registro de usuarios, en la que se encripte la contraseña antes de almacenarla:

```
const users = []; // Simula una base de datos

app.post('/register', async (req, res) => {
  const { username, password } = req.body;

  // Encriptar la contraseña
  const hashedPassword = await bcrypt.hash(password, 10);
  const newUser = { username, password: hashedPassword };
  users.push(newUser);

  res.status(201).json({ message: 'Usuario registrado exitosamente' });
});
```

Login de usuario y generación de JWT
En la ruta de login, compara la contraseña proporcionada con la encriptada en la base de datos. Si es válida, genera un token JWT:

```
app.post('/login', async (req, res) => {
  const { username, password } = req.body;
  const user = users.find(user => user.username === username);

  if (!user || !(await bcrypt.compare(password, user.password))) {
    return res.status(401).json({ message: 'Credenciales incorrectas' });
  }

  // Generar token JWT
  const token = jwt.sign({ username: user.username }, process.env.JWT_SECRET, {
expiresIn: '1h' });

  res.json({ token });
});
```

5. Proteger rutas con JWT
Para proteger una ruta, verifica el token JWT recibido en los encabezados:

```
const authenticateToken = (req, res, next) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];

  if (!token) return res.status(401).json({ message: 'Token requerido' });
```

```
  jwt.verify(token, process.env.JWT_SECRET, (err, user) => {
    if (err) return res.status(403).json({ message: 'Token inválido' });
    req.user = user;
    next();
  });
};

// Ruta protegida
app.get('/protected', authenticateToken, (req, res) => {
  res.json({ message: 'Acceso autorizado', user: req.user });
});
```

6. Inicia el servidor
Finalmente, inicia el servidor:

```
app.listen(PORT, () => {
  console.log(` Servidor en funcionamiento en http://localhost:${PORT}` );
});
```

Resumen del flujo

1. **Registro**: El usuario se registra y se guarda su contraseña encriptada.
2. **Login**: El usuario inicia sesión, se verifica la contraseña y se genera un JWT si es válida.
3. **Acceso a rutas protegidas**: El usuario incluye el token en el encabezado Authorization (Bearer <token>), y el servidor verifica su validez para autorizar el acceso a rutas.

Esta es una configuración básica de JWT con Express y Node.js que puedes extender según las necesidades de tu aplicación.