Conceptos clave de JWT:

- **Header:** Contiene el tipo de token (JWT) y el algoritmo de cifrado (ej., HS256).
- **Payload:** Contiene las "claims" o afirmaciones sobre la entidad (generalmente el usuario) y datos adicionales. Aquí podrías incluir el ID del usuario, su rol (admin, cliente, etc.) y la fecha de expiración del token.
- **Signature:** Se crea combinando el header, el payload, y una "clave secreta" (secret key) del servidor. Esto asegura la integridad del token y que no ha sido alterado.

Pasos para implementar JWT en tu API de productos:

1. Instalar las dependencias necesarias:

   Necesitarás express (si aún no lo tienes), jsonwebtoken para manejar los JWT y dotenv para gestionar tus variables de entorno de forma segura.

   Bash
   ```
   npm install express jsonwebtoken dotenv
   ```

2. Configurar variables de entorno (.env):

   Es crucial que tu clave secreta JWT no esté directamente en tu código. Crea un archivo .env en la raíz de tu proyecto:

   ```
   JWT_SECRET=tu_clave_secreta_super_segura_aqui
   ```

   Y carga estas variables al inicio de tu aplicación:

   JavaScript
   ```
   // server.js o app.js
   require('dotenv').config();
   const express = require('express');
   const jwt = require('jsonwebtoken');
   const app = express();

   app.use(express.json()); // Para parsear el body de las peticiones JSON
   ```

3. Implementar el registro y/o login de usuarios:

   Antes de poder proteger tus rutas de productos, necesitas una forma para que los usuarios se registren y/o inicien sesión y obtengan un token.

   - **Registro (ejemplo simplificado, sin base de datos):**
     JavaScript
     ```
     // Esto es solo un ejemplo. En un caso real, guardarías usuarios en una BD.
     const users = []; // Array para simular una base de datos de usuarios

     app.post('/register', (req, res) => {
         const { username, password } = req.body;
         // Aquí deberías hashear la contraseña antes de guardarla
         // Por simplicidad, no lo haremos en este ejemplo
         users.push({ username, password });
     ```

```
    res.status(201).send('Usuario registrado exitosamente');
});
```

- Login y generación de JWT:

  Cuando un usuario inicia sesión correctamente, le generas un token.

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

      // En un caso real, verificarías las credenciales contra tu base de datos
      const user = users.find(u => u.username === username && u.password
  === password);

      if (!user) {
          return res.status(401).send('Credenciales inválidas');
      }

      // Generar el token JWT
      const accessToken = jwt.sign(
          { username: user.username, role: 'user' }, // Payload: información que
  quieres guardar en el token
          process.env.JWT_SECRET,                    // Clave secreta
          { expiresIn: '1h' }                        // Opciones: expira en 1 hora
      );

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

4. Crear un middleware para verificar el JWT:

   Este middleware se ejecutará antes de tus rutas protegidas de productos para validar el token.

   JavaScript
   ```
   function authenticateToken(req, res, next) {
       const authHeader = req.headers['authorization'];
       const token = authHeader && authHeader.split(' ')[1]; // Formato: Bearer TOKEN

       if (token == null) {
           return res.status(401).send('No se proporcionó token'); // Unauthorized
       }

       jwt.verify(token, process.env.JWT_SECRET, (err, user) => {
           if (err) {
               return res.status(403).send('Token inválido o expirado'); // Forbidden
           }
           req.user = user; // Guarda la información del usuario decodificada en el
   request
           next(); // Pasa al siguiente middleware/ruta
       });
   }
   ```

5. Proteger tus rutas de productos:

Ahora, aplica el middleware authenticateToken a las rutas de tu API de productos que deseas proteger.

JavaScript

```
// Simulación de productos
const products = [
    { id: 1, name: 'Laptop', price: 1200 },
    { id: 2, name: 'Mouse', price: 25 },
    { id: 3, name: 'Teclado', price: 75 }
];

// Ruta para obtener todos los productos (protegida)
app.get('/products', authenticateToken, (req, res) => {
    // En este punto, req.user contiene la información del usuario del token
    console.log('Usuario autenticado:', req.user);
    res.json(products);
});

// Ruta para agregar un producto (ejemplo, podría requerir un rol específico)
app.post('/products', authenticateToken, (req, res) => {
    // Aquí podrías agregar lógica para verificar roles, por ejemplo:
    if (req.user.role !== 'admin') {
        return res.status(403).send('Acceso denegado. Se requiere rol de
administrador.');
    }
    const newProduct = req.body;
    products.push(newProduct);
    res.status(201).json(newProduct);
});

// Iniciar el servidor
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
    console.log(`Servidor corriendo en el puerto ${PORT}`);
});
```

Cómo probarlo (usando Postman o similar):

1. **Registrar un usuario (si es necesario):**
    - POST a http://localhost:3000/register
    - Body (JSON): { "username": "testuser", "password": "password123" }
2. **Iniciar sesión para obtener el token:**
    - POST a http://localhost:3000/login
    - Body (JSON): { "username": "testuser", "password": "password123" }
    - La respuesta te devolverá un accessToken. Copia este token.
3. **Acceder a una ruta protegida (ej. /products):**
    - GET a http://localhost:3000/products
    - En los "Headers" de la petición, agrega:
        - Key: Authorization
        - Value: Bearer TU_TOKEN_COPIADO (reemplaza TU_TOKEN_COPIADO con el token real)

Si el token es válido, deberías recibir la lista de productos. Si el token está ausente, inválido o expirado, recibirás un error 401 o 403.

Consideraciones adicionales:

- **Hasheo de contraseñas:** Siempre utiliza librerías como `bcrypt.js` para hashear las contraseñas antes de almacenarlas en la base de datos y para verificarlas durante el login. Nunca guardes contraseñas en texto plano.
- **Refresh Tokens:** Para una mejor experiencia de usuario y seguridad, considera implementar refresh tokens. Los access tokens son de corta duración, y los refresh tokens (más largos y almacenados de forma segura) se usan para obtener nuevos access tokens sin que el usuario tenga que iniciar sesión de nuevo.
- **Roles y permisos:** Puedes expandir el payload del JWT para incluir roles de usuario y luego usar esos roles en tus middlewares para implementar autorización basada en roles (ej., solo los administradores pueden crear productos).
- **Almacenamiento del token en el cliente:** Evita almacenar JWT en `localStorage` o `sessionStorage` directamente debido a vulnerabilidades XSS. Es preferible usar `HttpOnly` cookies.
- **Error Handling:** Implementa un manejo de errores robusto para todas tus rutas.