

*“AÑO DE LA RECUPERACIÓN DE LA ECONOMÍA PERUANA”*

**UNIVERSIDAD NACIONAL DEL CENTRO DEL PERÚ**



**FACULTAD DE INGENIERÍA DE SISTEMAS**

**INFORME - PROYECTO JWT**

**DOCENTE:**

SUASNABAR TERREL, Jaime

**PRESENTADO POR:**

ESPINOZA MORALES, Keyla Xiomara

LEON GARCIA, David Daniel

**HUANCAYO - PERÚ**

**2025**

## AUTENTICACIÓN

La autenticación es un proceso fundamental en ciberseguridad que permite verificar la identidad de usuarios, servicios y aplicaciones antes de concederles acceso a los recursos de una organización. Su propósito es garantizar que solo personas con los permisos adecuados puedan interactuar con los sistemas, reduciendo así el riesgo de acceso no autorizado y posibles ataques.

Este procedimiento se compone de tres fases esenciales

- **Identificación:** El usuario declara quién es, generalmente con un nombre de usuario.
- **Autenticación:** Se verifica la identidad mediante una contraseña, un dispositivo de tokens o biometría.
- **Autorización:** Se comprueba que el usuario tiene los permisos adecuados para acceder.

La autenticación protege datos personales, redes y aplicaciones contra ataques. Si los mecanismos son deficientes, los infiltrados pueden obtener credenciales, lo que conlleva riesgos como el robo de datos, la instalación de malware y el incumplimiento de regulaciones de privacidad.

### Tipos de Autenticación

Microsoft define 8 tipos de autenticación.

- **Autenticación basada en contraseña:** Método tradicional en el que los usuarios crean contraseñas seguras combinando números, letras y símbolos. No obstante, el robo de contraseñas es común, por lo que muchas organizaciones están adoptando alternativas más seguras.
- **Autenticación basada en certificado:** Utiliza certificados digitales para autenticar personas o dispositivos, mejorando la seguridad mediante cifrado.
- **Autenticación biométrica:** Se basa en características físicas únicas del usuario, como la huella dactilar o el reconocimiento facial. Este método ofrece mayor seguridad y comodidad, ya que no requiere recordar contraseñas.

- **Autenticación basada en tokens:** Emplea códigos generados temporalmente por dispositivos de seguridad, verificando la identidad del usuario mediante un proceso dinámico.
- **Contraseña de un solo uso (OTP):** Se envía un código temporal al usuario vía SMS, correo electrónico o hardware especializado, asegurando que el acceso sea válido solo por un breve período.
- **Notificación push:** Requiere que el usuario apruebe manualmente un acceso desde su dispositivo móvil.
- **Autenticación por voz:** Consiste en la verificación de identidad mediante una llamada telefónica que solicita la confirmación verbal o el ingreso de un código.
- **Autenticación multifactor:** Método que combina dos o más mecanismos de seguridad, como una contraseña junto con una OTP o biometría. Es considerado uno de los enfoques más sólidos para evitar accesos no autorizados.

## TOKEN

En el ámbito de la autenticación y autorización, un token es un objeto digital que contiene información sobre la identidad del solicitante y los permisos que posee para acceder a determinados recursos. En la mayoría de los procesos de autenticación, la aplicación intercambia una credencial por un token, el cual posteriormente determina los niveles de acceso permitidos (Google Cloud).

### Tipo de Token

- **Token de acceso**
  - Se usa para autorizar acceso a un sistema o servicio.
  - Permite realizar acciones dentro de una aplicación sin necesidad de ingresar usuario y contraseña en cada solicitud.
  - Ejemplo: Cuando inicias sesión en un sitio web y puedes navegar sin repetir la autenticación.
- **Token de ID**
  - Contiene información sobre la identidad del usuario.
  - Se usa en autenticación para confirmar quién eres.

- Ejemplo: Al iniciar sesión con Google, el sistema usa un token de ID para saber quién está accediendo.
- **Token de actualización**
  - Permite renovar un token de acceso vencido sin que el usuario tenga que autenticarse de nuevo.
  - Se usa para mantener sesiones activas sin repetir inicios de sesión.
  - Ejemplo: Cuando una aplicación renueva automáticamente tu sesión sin que tengas que volver a iniciar sesión.
- **Token del portador**
  - Funciona como un pase de acceso: si alguien tiene el token, puede usarlo para acceder a recursos sin necesidad de contraseña.
  - Debe manejarse con seguridad para evitar accesos no autorizados.
  - Ejemplo: Al usar una API, si tienes un token del portador válido, puedes hacer solicitudes sin repetir autenticación.

## **JSON WEB TOKEN**

IBM (2025) define JWT son un estándar abierto definido en la RFC 7519, que permite representar de manera segura reclamaciones entre dos partes. Estas reclamaciones pueden estar relacionadas con distintos procesos comerciales, aunque su uso más común es en la autenticación de usuarios. Por ejemplo, un JWT puede representar la identidad de un usuario y sus permisos dentro de un sistema, permitiendo su acceso sin necesidad de reenviar credenciales en cada solicitud.

### **Características y ventajas de los JWT**

El uso de JWT proporciona múltiples beneficios:

- Ligereza y facilidad de uso, especialmente en aplicaciones cliente como plataformas móviles.
- Autonomía, lo que permite que el servidor consuma directamente el token sin necesidad de consultar bases de datos adicionales.
- Firma digital, utilizando algoritmos como HMAC-SHA256 o RSA-SHA256, lo que garantiza la autenticidad del token.
- Expiración incorporada, estableciendo un período de validez tras el cual el token debe ser renovado.

- Extensibilidad, ya que permite incluir reclamaciones personalizadas según las necesidades de la aplicación.
- Compatibilidad con sistemas de inicio de sesión único (SSO), como OpenID Connect.

## Estructura de un JWT

Un JWT consta de tres partes fundamentales:

1. **Encabezado:** Indica el tipo de token (JWT) y el algoritmo de firma utilizado (HMAC-SHA256, RSA-SHA256, etc.). Se codifica en Base64Url.
2. **Carga útil:** Contiene las reclamaciones, entre ellas:
  - **iss** (emisor del token)
  - **exp** (fecha de expiración)
  - **sub** (identidad del usuario)
  - **aud** (audiencia permitida) También puede incluir atributos personalizados, como la función o grupo del usuario.
3. **Firma:** Se genera al combinar el encabezado y la carga útil codificados con la clave de firma especificada. Su propósito es validar la autenticidad del JWT y evitar modificaciones malintencionadas.

## Flujo de autenticación con JWT

El proceso de autenticación mediante JWT sigue estos pasos:

1. El usuario ingresa sus credenciales y solicita acceso.
2. Tras la validación, el servidor genera un JWT y lo entrega al usuario.
3. En cada solicitud posterior, el cliente envía el JWT en la cabecera Authorization.
4. El servidor valida el JWT para identificar al usuario y conceder el acceso a los recursos solicitados.

## POSTMAN

Postman es una herramienta diseñada para facilitar la prueba y gestión de APIs REST, permitiendo a desarrolladores y testers realizar solicitudes de manera eficiente. Inicialmente, Postman surgió como una extensión para el navegador Google Chrome, pero con el tiempo evolucionó hasta convertirse en una aplicación independiente con soporte para Windows, Mac y Linux.

### Funciones principales de Postman

Postman es ampliamente utilizado en el desarrollo de software debido a sus múltiples funcionalidades, entre las que destacan:

- **Pruebas de APIs:** Permite realizar solicitudes HTTP para verificar el funcionamiento de servicios web.
- **Organización de colecciones:** Facilita la estructuración de pruebas en carpetas y módulos.
- **Gestión del ciclo de vida de APIs:** Desde la conceptualización hasta el mantenimiento.
- **Generación de documentación:** Automatiza la creación de documentación técnica sobre APIs.
- **Trabajo con entornos:** Permite gestionar configuraciones para desarrollo, pruebas y producción.

### Ventajas de Postman

- Gran comunidad de usuarios.
- Interfaz intuitiva y atractiva.
- Posibilidad de colaboración en equipo.
- Integración con otras herramientas.
- Soporte para scripts en JavaScript.
- Gestión eficiente de colecciones de pruebas.

## **APLICACIÓN**

El proyecto se llama Mariscal-Auth, contiene una interfaz gráfica para loguearse solo los administradores el cual debe contener los campos user y password además si un usuario ingresa el usuario o contraseña incorrecta, entonces sale un mensaje indicando credenciales incorrectas.

Si el usuario ingresa credenciales correctas, entonces se direcciona a una página siguiente en la cual haya un menú donde se pueda agregar estudiantes en los 5 diferentes grados y correspondientes secciones que son solo la A,B y la C.

### **Stack del proyecto**

#### **1. Frontend**

El frontend representa la parte visible de la aplicación, donde los usuarios interactúan a través del navegador. Se utilizan las siguientes tecnologías:

- **HTML (HyperText Markup Language):** Se encarga de la estructura y organización del contenido de las páginas web. En este caso, archivos como index.html y dashboard.html definen las vistas principales del proyecto.
- **CSS (Cascading Style Sheets):** Se utiliza para diseñar y estilizar la presentación visual de la aplicación. El archivo styles.css contiene las reglas de diseño que dan forma a la interfaz.
- **JavaScript (Vanilla JS):** Se usa para implementar la lógica del lado cliente, permitiendo interacciones dinámicas dentro de la aplicación. El archivo script.js contiene funciones de manipulación del DOM y eventos de usuario.

#### **2. Backend**

El backend gestiona las operaciones internas del sistema, procesando solicitudes y respondiendo con datos apropiados. Se han utilizado las siguientes herramientas:

- **Node.js:** Un entorno de ejecución de JavaScript en el servidor, que permite manejar peticiones de manera eficiente. En este proyecto, el archivo principal index.js ejecuta el código backend.

- Express.js: Un framework de Node.js diseñado para gestionar rutas y construir APIs REST. Se implementa en archivos como routes/auth.js y routes/students.js para manejar la autenticación y la gestión de usuarios.

### **3. Base de datos**

La base de datos mantiene los datos persistentes y permite su consulta y modificación mediante el backend. Para ello, se usa:

- SQLite: Una base de datos liviana y embebida que permite almacenar información sin necesidad de un servidor externo. El archivo mariscal\_castilla.db contiene los datos del proyecto.
- sqlite3: Librería utilizada en Node.js para conectarse a la base de datos SQLite y ejecutar consultas. Se implementa en db.js para gestionar la interacción con la base de datos.

### **4. Seguridad / Autenticación**

La autenticación y la protección de datos son esenciales en cualquier aplicación. Se han utilizado las siguientes tecnologías para garantizar la seguridad:

- JWT (JSON Web Tokens): Un método de autenticación basado en tokens, permitiendo la gestión segura de sesiones de usuario. La librería jsonwebtoken se usa en auth.js para generar y validar tokens.
- bcryptjs: Se utiliza para encriptar contraseñas y almacenarlas de manera segura en la base de datos, evitando el almacenamiento en texto plano y reduciendo el riesgo de vulneraciones.

### **Guía de Aplicación**

Indica los pasos a realizar de cómo crear un sistema de autenticación con Json Web Token en la API POSTMAN, todo esto en la base de datos SQLite, estableciendo rutas de acceso para así registrar a los usuarios, seguidamente logueandolos para finalmente autenticarlos con el token generado por POSTMAN.

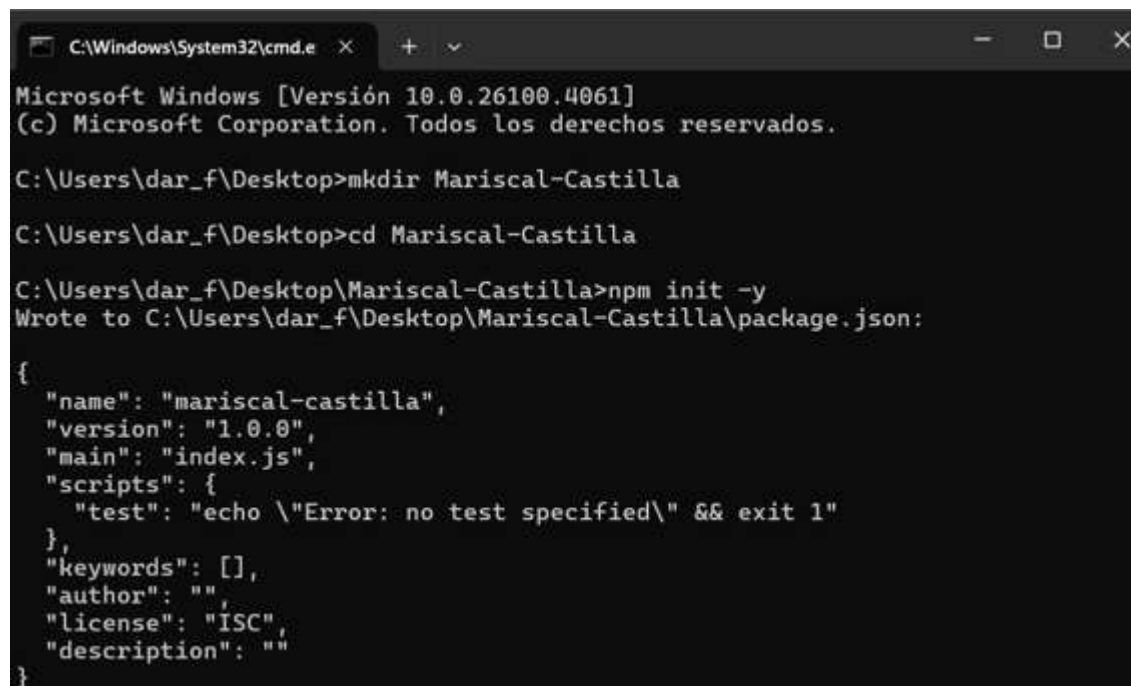


## Paso 01 – Instalación de Herramientas Básicas

Se necesita tener instalado Node.js (incluye npm que es el gestor de paquetes de Node).

## Paso 02 – Crear el Proyecto en NODE.JS

Primero se crea la carpeta en el escritorio con el comando: `mkdir David-Auth`, posteriormente abrimos la carpeta con el comando: `cd David-Auth`, luego introducimos el comando: `npm init -y`, esto crea un archivo `package.json` que es la configuración del proyecto.



```
C:\Windows\System32\cmd.e X + v
Microsoft Windows [Versión 10.0.26100.4061]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\dar_f\Desktop>mkdir Mariscal-Castilla

C:\Users\dar_f\Desktop>cd Mariscal-Castilla

C:\Users\dar_f\Desktop\Mariscal-Castilla>npm init -y
Wrote to C:\Users\dar_f\Desktop\Mariscal-Castilla\package.json:

{
  "name": "mariscal-castilla",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}
```

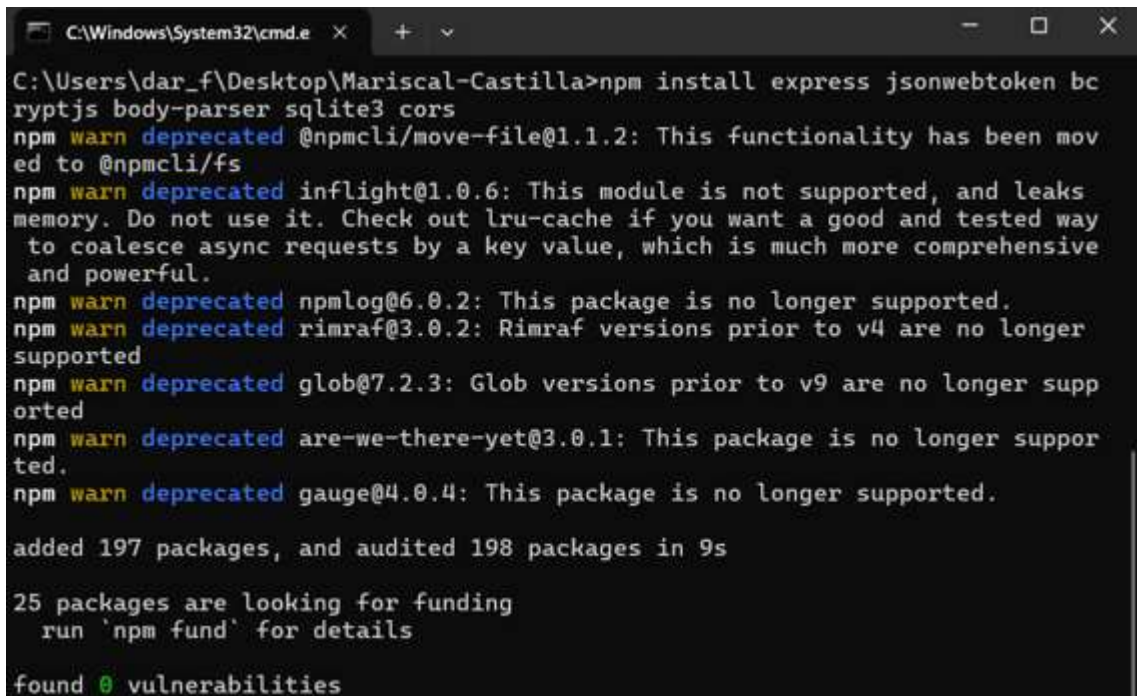
## Paso 03 – Instalar Dependencias

Se introduce el siguiente comando: `npm install express jsonwebtoken bcrypt sqlite3 body-parser cors`.

- Express: Framework para crear el servidor web.
- Jsonwebtoken: Librería para crear y validar JWT.
- Bcrypt: Para cifrar contraseñas.
- Sqlite3: Base de datos local.
- Body-parser: Para procesar los datos enviados por el usuario.

- Cors: Permite que puedas consumir tu API desde otras apps (opcional pero recomendado).

Luego en el mismo terminal: “echo > index.js” y “echo > database.js”, esto creará dos archivos dentro de la carpeta David-Auth, la primera carpeta se llamará index.js y la segunda database.js



```
C:\Windows\System32\cmd.e X + v
C:\Users\dar_f\Desktop\Mariscal-Castilla>npm install express jsonwebtoken bcryptjs body-parser sqlite3 cors
npm warn deprecated @npmcli/move-file@1.1.2: This functionality has been moved to @npmcli/fs
npm warn deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out lru-cache if you want a good and tested way to coalesce async requests by a key value, which is much more comprehensive and powerful.
npm warn deprecated npmlog@6.0.2: This package is no longer supported.
npm warn deprecated rimraf@3.0.2: Rimraf versions prior to v4 are no longer supported
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported
npm warn deprecated are-we-there-yet@3.0.1: This package is no longer supported.
npm warn deprecated gauge@4.0.4: This package is no longer supported.

added 197 packages, and audited 198 packages in 9s

25 packages are looking for funding
  run 'npm fund' for details

found 0 vulnerabilities
```

#### Paso 04 – Dashboard.html

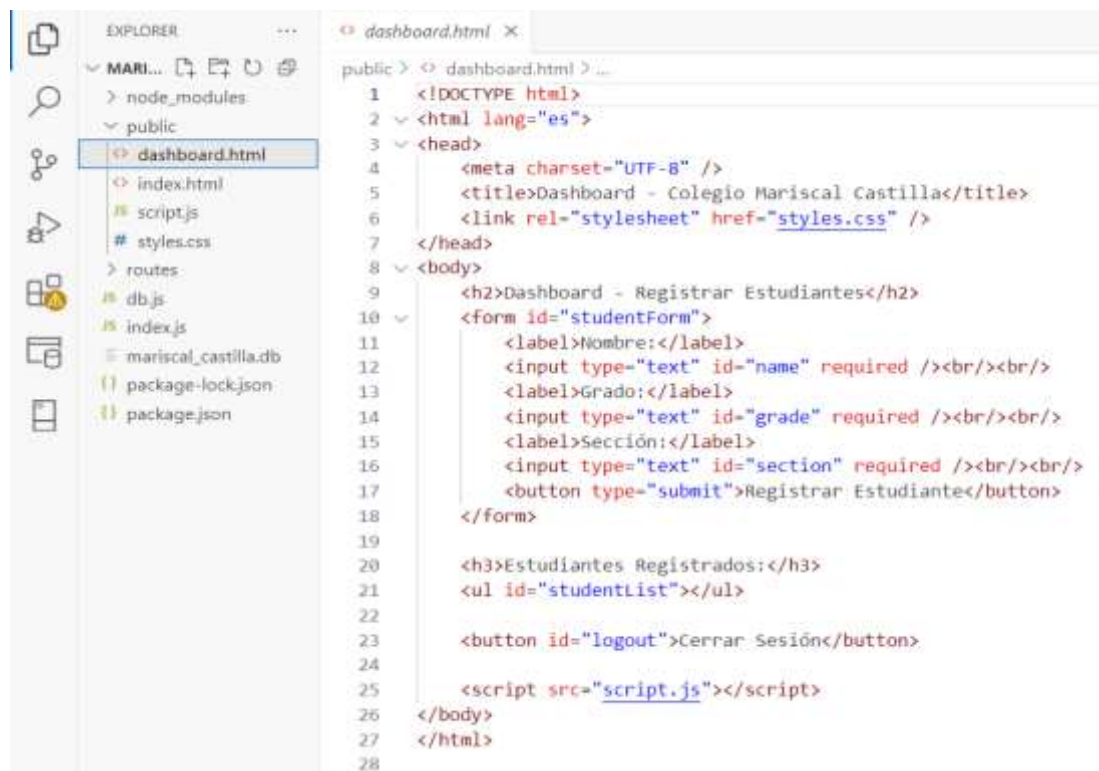
El archivo dashboard.html define la interfaz de usuario del sistema de gestión de estudiantes para el Colegio Mariscal Castilla. La estructura incluye un título principal que identifica la plataforma y proporciona un formulario para el registro de estudiantes. Dentro del formulario, se han incorporado campos de entrada para que el usuario ingrese el nombre, grado y sección del estudiante, así como un botón para enviar la información.

Una vez que los datos son ingresados en el formulario, estos se muestran dinámicamente en una sección específica destinada a los estudiantes registrados. La información es presentada dentro de un elemento <ul> (lista no ordenada), lo que facilita la visualización clara y estructurada de los registros existentes.

Para mejorar la funcionalidad y el diseño de la página, dashboard.html importa dos archivos externos:

- styles.css: Define la apariencia visual de la interfaz, incluyendo colores, márgenes y organización de los elementos.
- script.js: Contiene la lógica necesaria para capturar los datos del formulario y agregarlos dinámicamente a la lista de estudiantes registrados.

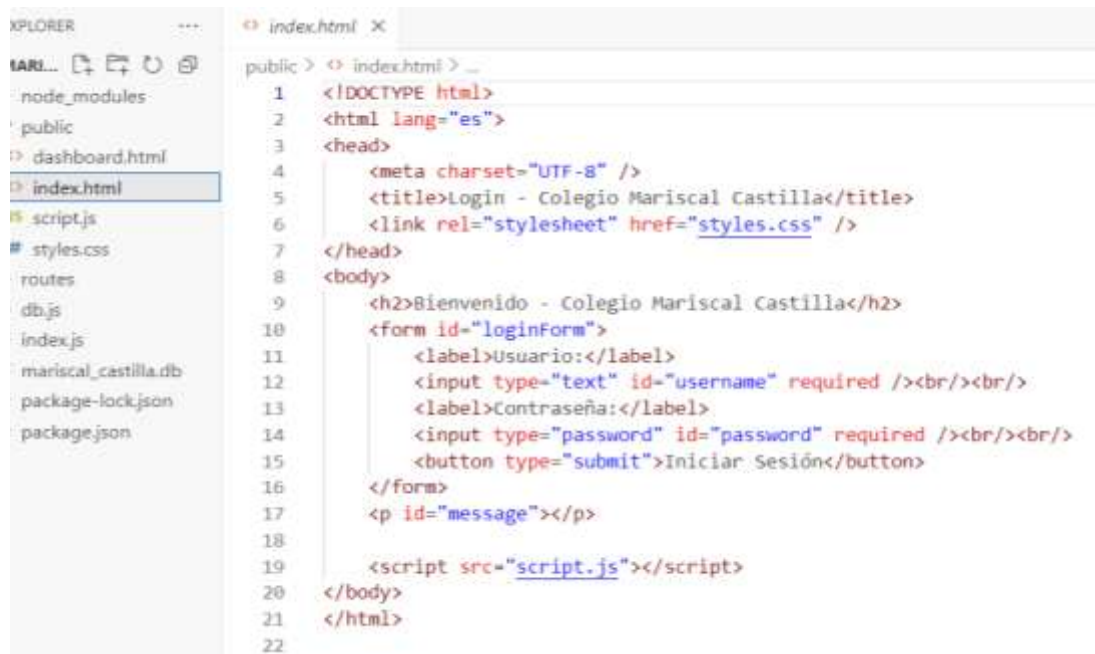
El sistema incluye un mecanismo de seguridad mediante un botón de cierre de sesión (Cerrar Sesión). Este elemento permite que el usuario cierre la sesión de manera segura, eliminando la información de autenticación almacenada y evitando accesos no autorizados.



```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8" />
5   <title>Dashboard - Colegio Mariscal Castilla</title>
6   <link rel="stylesheet" href="styles.css" />
7 </head>
8 <body>
9   <h2>Dashboard - Registrar Estudiantes</h2>
10  <form id="studentForm">
11    <label>Nombre:</label>
12    <input type="text" id="name" required /><br/><br/>
13    <label>Grado:</label>
14    <input type="text" id="grade" required /><br/><br/>
15    <label>Sección:</label>
16    <input type="text" id="section" required /><br/><br/>
17    <button type="submit">Registrar Estudiante</button>
18  </form>
19
20  <h3>Estudiantes Registrados:</h3>
21  <ul id="studentList"></ul>
22
23  <button id="logout">Cerrar Sesión</button>
24
25  <script src="script.js"></script>
26 </body>
27 </html>
28
```

## Paso 05 – Index.html

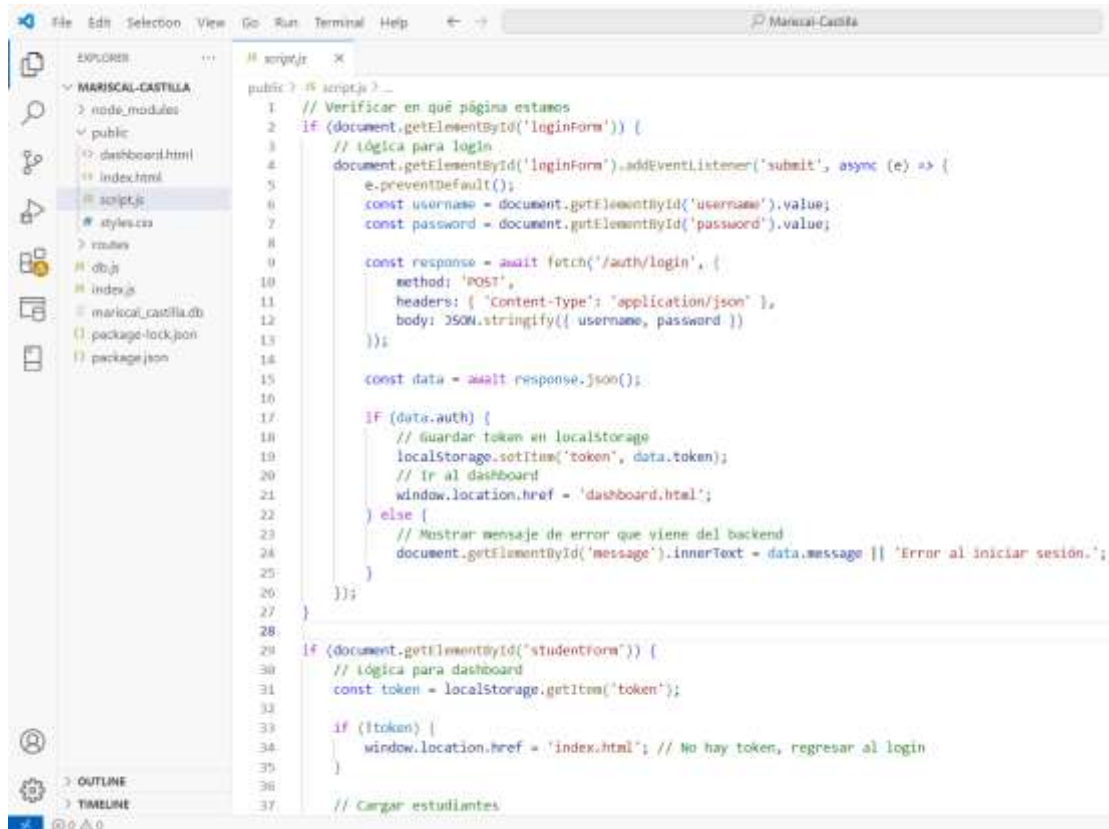
El código HTML de la página de inicio de sesión para el Colegio Mariscal Castilla contiene un formulario con campos para ingresar usuario y contraseña, ambos obligatorios, junto con un botón para iniciar sesión. Se vinculan una hoja de estilos externa (styles.css) y un archivo de script (script.js), lo que permite mejorar la apariencia y agregar funcionalidad dinámica. Este diseño proporciona una estructura básica para la autenticación de usuarios en la plataforma web.



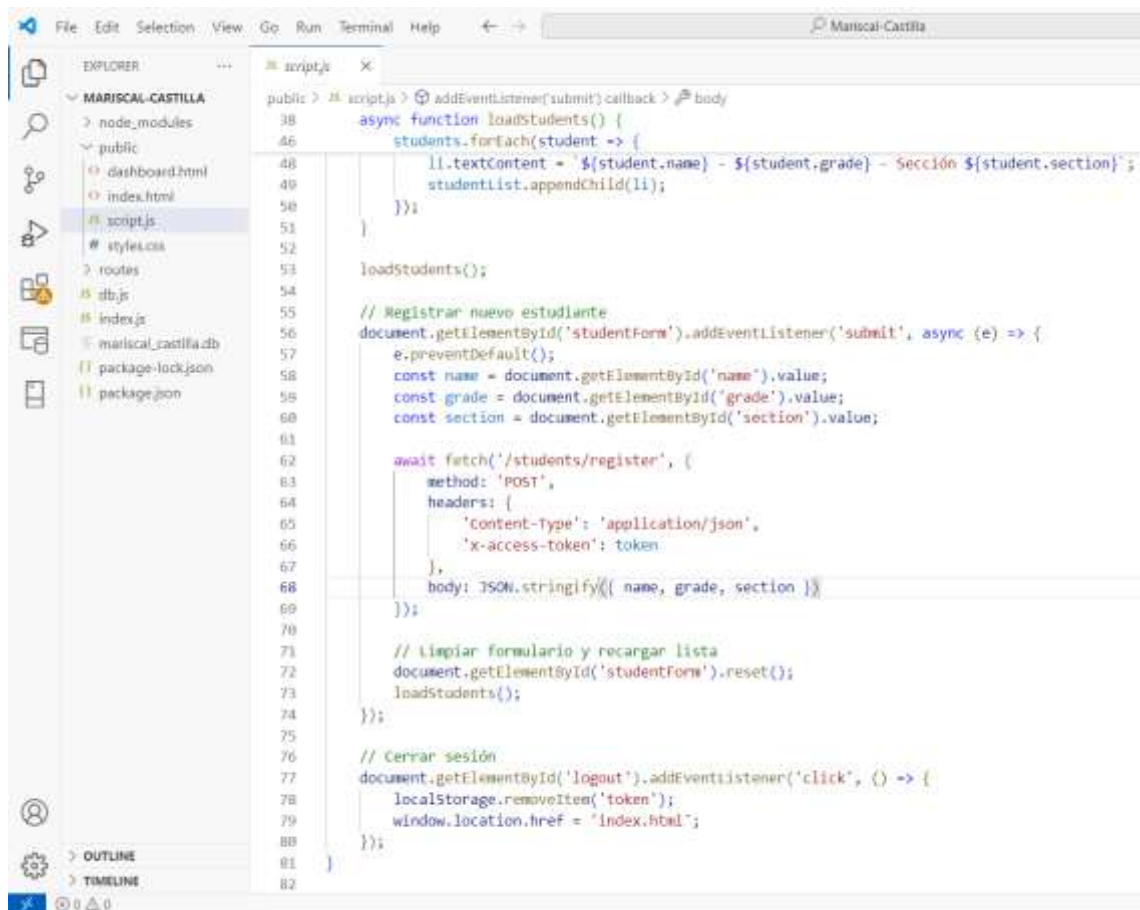
```
public > <> index.html > ...
1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4      <meta charset="UTF-8" />
5      <title>Login - Colegio Mariscal Castilla</title>
6      <link rel="stylesheet" href="styles.css" />
7  </head>
8  <body>
9      <h2>Bienvenido - Colegio Mariscal Castilla</h2>
10     <form id="loginForm">
11         <label>Usuario:</label>
12         <input type="text" id="username" required /><br/><br/>
13         <label>Contraseña:</label>
14         <input type="password" id="password" required /><br/><br/>
15         <button type="submit">Iniciar Sesión</button>
16     </form>
17     <p id="message"></p>
18
19     <script src="script.js"></script>
20 </body>
21 </html>
22
```

## Paso 06 – Scrip.js

En las dos siguiente imágenes se muestra que el archivo script.js gestiona la lógica de autenticación y navegación en la aplicación. En la sección de login, captura los valores de usuario y contraseña, los envía al backend mediante una solicitud POST y, si la autenticación es exitosa, almacena el token JWT en localStorage para mantener la sesión activa y redirige al usuario al dashboard.html. Para el dashboard, valida la presencia del token en localStorage y, si no existe, redirige al usuario al login, evitando accesos no autorizados.



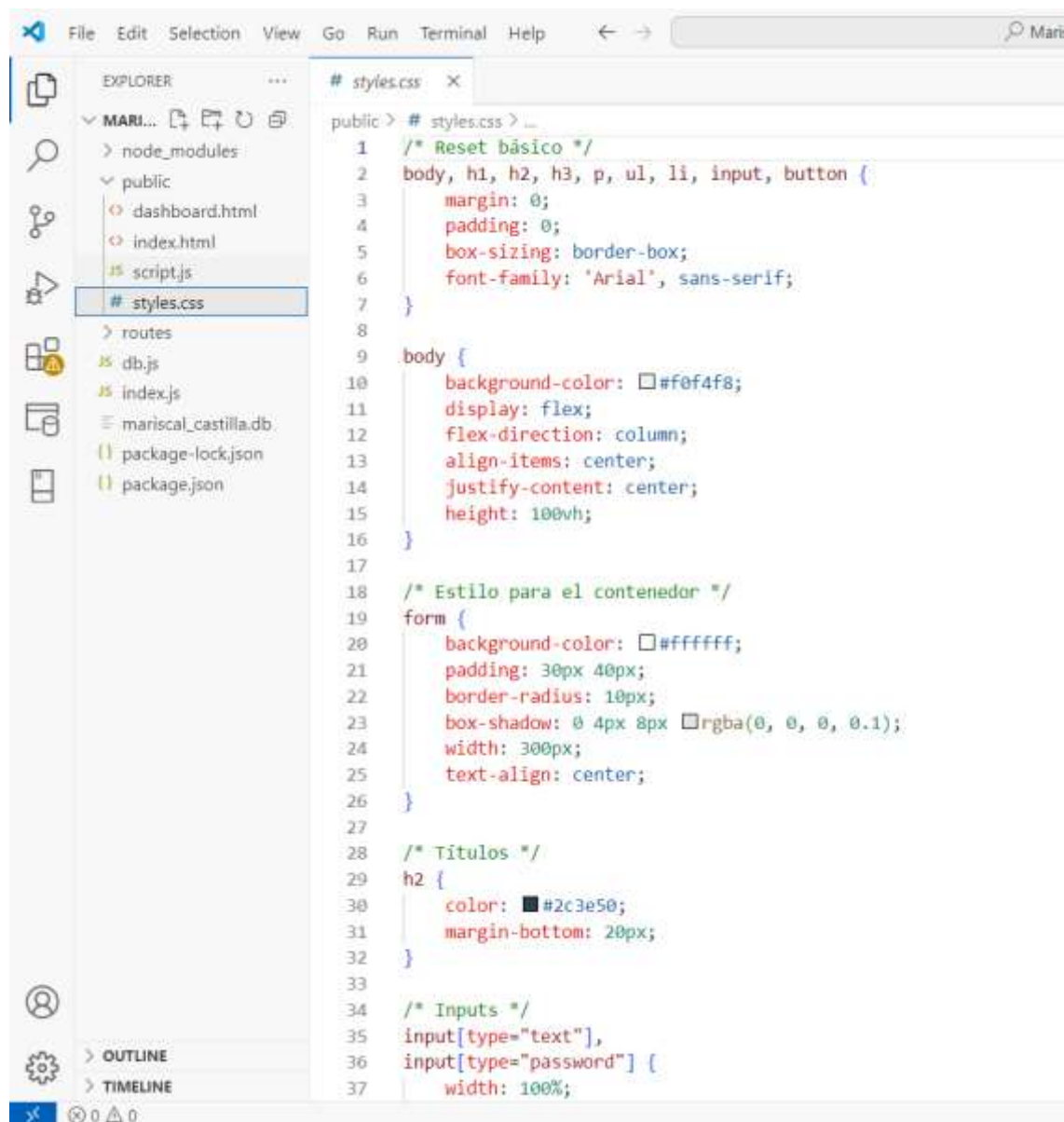
```
public? script.js
1 // Verificar en qué página estamos
2 if (document.getElementById('loginForm')) {
3   // lógica para login
4   document.getElementById('loginForm').addEventListener('submit', async (e) => {
5     e.preventDefault();
6     const username = document.getElementById('username').value;
7     const password = document.getElementById('password').value;
8
9     const response = await fetch('/auth/login', {
10       method: 'POST',
11       headers: { 'Content-Type': 'application/json' },
12       body: JSON.stringify({ username, password })
13     });
14
15     const data = await response.json();
16
17     if (data.auth) {
18       // Guardar token en localStorage
19       localStorage.setItem('token', data.token);
20       // Ir al dashboard
21       window.location.href = 'dashboard.html';
22     } else {
23       // Mostrar mensaje de error que viene del backend
24       document.getElementById('message').innerText = data.message || 'Error al iniciar sesión.';
25     }
26   });
27 }
28
29 if (document.getElementById('studentForm')) {
30   // lógica para dashboard
31   const token = localStorage.getItem('token');
32
33   if (!token) {
34     window.location.href = 'index.html'; // No hay token, regresar al login
35   }
36
37   // Cargar estudiantes
```



```
public? script.js
38 // addEventListener('submit') callback > body
39 async function loadStudents() {
40   students.forEach(student => {
41     li.textContent = `${student.name} - ${student.grade} - Sección ${student.section}`;
42     studentList.appendChild(li);
43   });
44 }
45
46 loadStudents();
47
48 // Registrar nuevo estudiante
49 document.getElementById('studentForm').addEventListener('submit', async (e) => {
50   e.preventDefault();
51   const name = document.getElementById('name').value;
52   const grade = document.getElementById('grade').value;
53   const section = document.getElementById('section').value;
54
55   await fetch('/students/register', {
56     method: 'POST',
57     headers: {
58       'Content-Type': 'application/json',
59       'x-access-token': token
60     },
61     body: JSON.stringify({ name, grade, section })
62   });
63
64   // Limpiar formulario y recargar lista
65   document.getElementById('studentForm').reset();
66   loadStudents();
67 });
68
69 // Cerrar sesión
70 document.getElementById('logout').addEventListener('click', () => {
71   localStorage.removeItem('token');
72   window.location.href = 'index.html';
73 });
74
75 }
76
77
78
79
80
81
82
```

## Paso 07 – Styles.css

El archivo styles.css define la apariencia visual de la aplicación, estableciendo estilos para el formulario, los títulos y los campos de entrada. Se aplica un reset básico para asegurar una consistencia en el diseño, configurando márgenes, paddings y fuentes predeterminadas. El fondo del cuerpo se establece con un color claro y una disposición centrada para mejorar la presentación. Se estiliza el contenedor del formulario, añadiendo bordes redondeados, sombreado y espaciado adecuado para mejorar la experiencia del usuario. Además, los títulos reciben un color distintivo y se añade formato a los inputs, asegurando que ocupen el ancho completo disponible.

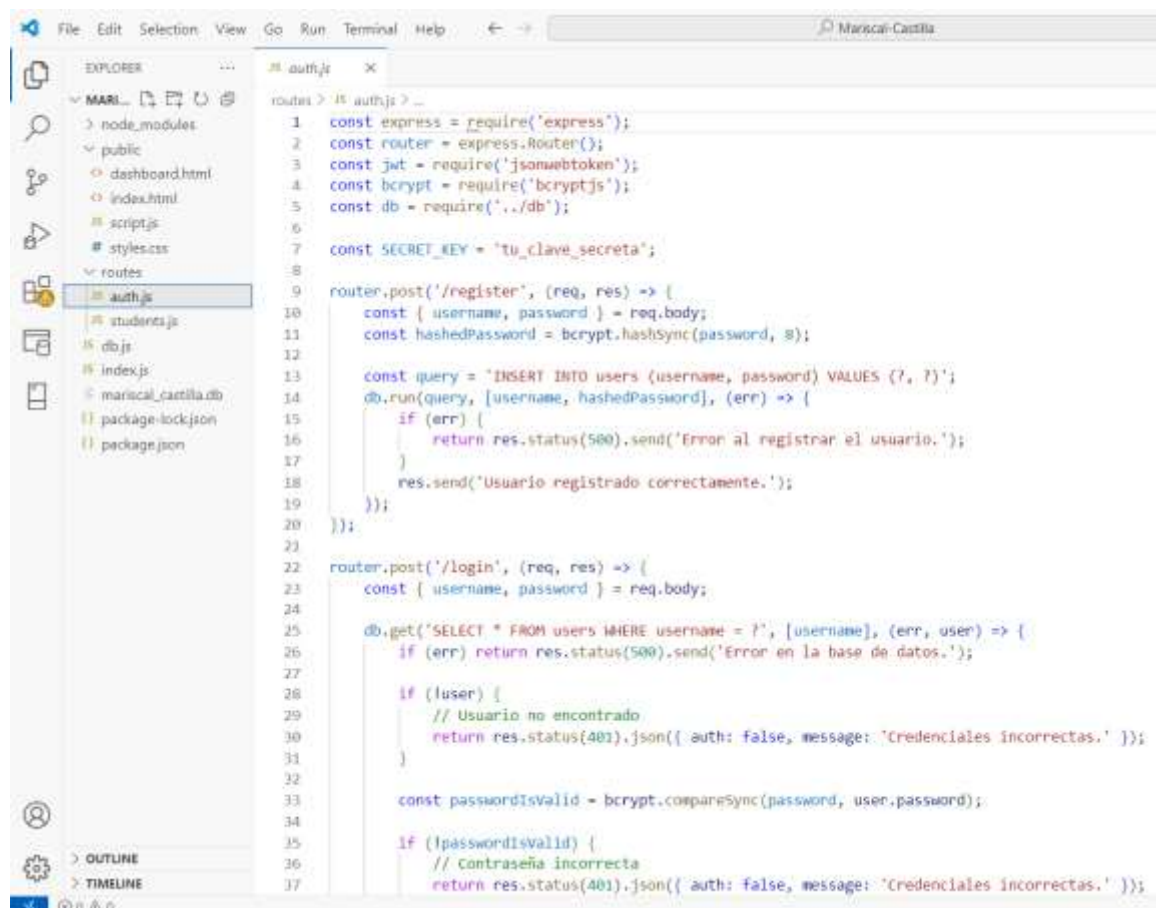


```
1  /* Reset básico */
2  body, h1, h2, h3, p, ul, li, input, button {
3      margin: 0;
4      padding: 0;
5      box-sizing: border-box;
6      font-family: 'Arial', sans-serif;
7  }
8
9  body {
10     background-color: #f0f4f8;
11     display: flex;
12     flex-direction: column;
13     align-items: center;
14     justify-content: center;
15     height: 100vh;
16 }
17
18 /* Estilo para el contenedor */
19 form {
20     background-color: #ffffff;
21     padding: 30px 40px;
22     border-radius: 10px;
23     box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
24     width: 300px;
25     text-align: center;
26 }
27
28 /* Títulos */
29 h2 {
30     color: #2c3e50;
31     margin-bottom: 20px;
32 }
33
34 /* Inputs */
35 input[type="text"],
36 input[type="password"] {
37     width: 100%;
```



## Paso 08 – Auth.js

El archivo auth.js gestiona la autenticación de usuarios en el sistema mediante Express, implementando rutas para registro y inicio de sesión. Se usa bcryptjs para cifrar contraseñas antes de almacenarlas en SQLite, asegurando protección contra accesos no autorizados. En el proceso de login, el sistema verifica las credenciales, comparando la contraseña ingresada con la almacenada. Si son correctas, se genera un token JWT, permitiendo el acceso a recursos protegidos.

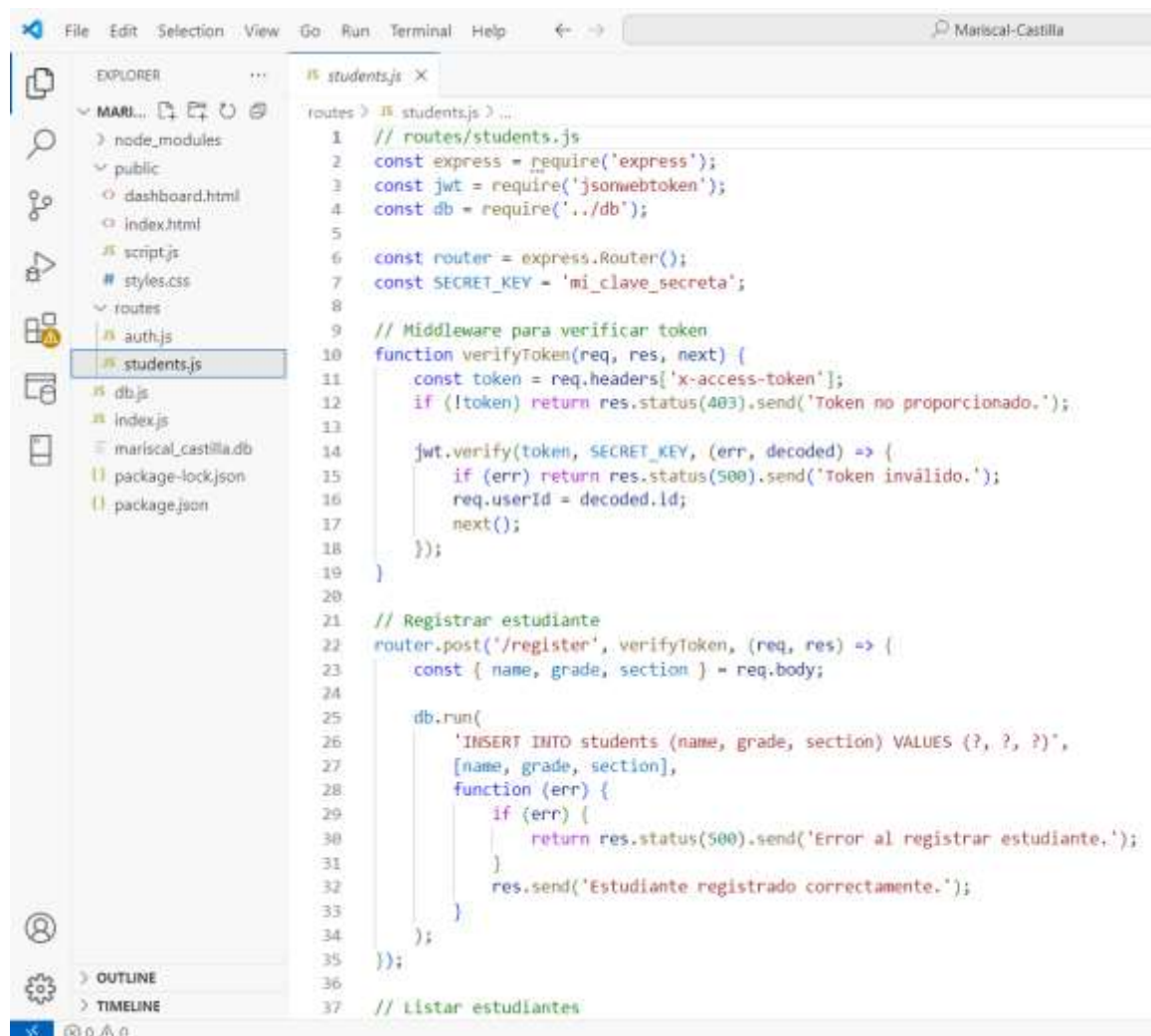


```
1 const express = require('express');
2 const router = express.Router();
3 const jwt = require('jsonwebtoken');
4 const bcrypt = require('bcryptjs');
5 const db = require('../db');
6
7 const SECRET_KEY = 'tu_clave_secreta';
8
9 router.post('/register', (req, res) => {
10   const { username, password } = req.body;
11   const hashedPassword = bcrypt.hashSync(password, 8);
12
13   const query = 'INSERT INTO users (username, password) VALUES (?, ?)';
14   db.run(query, [username, hashedPassword], (err) => {
15     if (err) {
16       return res.status(500).send('Error al registrar el usuario.');
```

## Paso 09 – Students.js

El archivo students.js gestiona las rutas relacionadas con la administración de estudiantes en una aplicación basada en Express.js. Se implementa un middleware de verificación de token para garantizar la seguridad, validando que el usuario tenga una sesión activa mediante JWT (JSON Web Token) antes de

acceder a los recursos protegidos. Se define una ruta POST para el registro de estudiantes, almacenando datos como nombre, grado y sección en una base de datos SQLite mediante consultas preparadas.



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays a project structure with folders like 'node\_modules', 'public', and 'routes'. The 'routes' folder is expanded, showing 'auth.js' and 'students.js'. The 'students.js' file is selected and its content is displayed in the main editor. The code defines an Express router for student management, including a token verification middleware and a POST endpoint for registering students.

```
1 // routes/students.js
2 const express = require('express');
3 const jwt = require('jsonwebtoken');
4 const db = require('../db');
5
6 const router = express.Router();
7 const SECRET_KEY = 'mi_clave_secreta';
8
9 // Middleware para verificar token
10 function verifyToken(req, res, next) {
11   const token = req.headers['x-access-token'];
12   if (!token) return res.status(403).send('Token no proporcionado.');
```

```
13
14   jwt.verify(token, SECRET_KEY, (err, decoded) => {
15     if (err) return res.status(500).send('Token inválido.');
```

```
16     req.userId = decoded.id;
17     next();
18   });
19 }
20
21 // Registrar estudiante
22 router.post('/register', verifyToken, (req, res) => {
23   const { name, grade, section } = req.body;
24
25   db.run(
26     'INSERT INTO students (name, grade, section) VALUES (?, ?, ?)',
27     [name, grade, section],
28     function (err) {
29       if (err) {
30         return res.status(500).send('Error al registrar estudiante.');
```

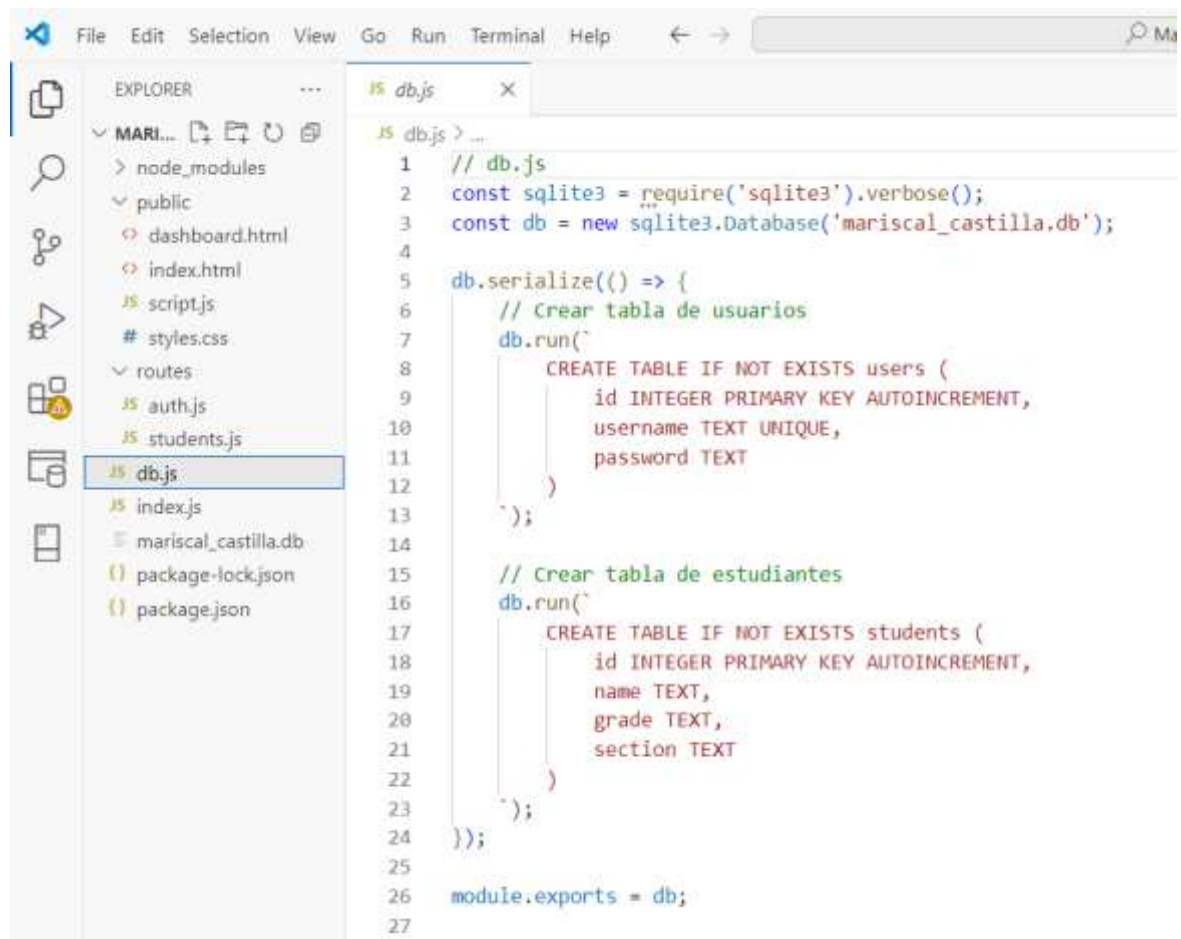
```
31       }
32       res.send('Estudiante registrado correctamente.');
```

```
33     }
34   );
35 });
36
37 // Listar estudiantes
```

## Paso 10 – db.js

El archivo db.js configura la base de datos SQLite utilizada en el sistema, creando de forma estructurada las tablas necesarias para gestionar usuarios y estudiantes. Se importa la librería sqlite3 y se establece la conexión con la base de datos mariscal\_castilla.db. Luego, mediante `serialize()`, se ejecutan consultas para crear la tabla `users`, que almacena `id`, `username` y `password`, y la tabla `students`, que registra `id`, `name`, `grade` y `section`.

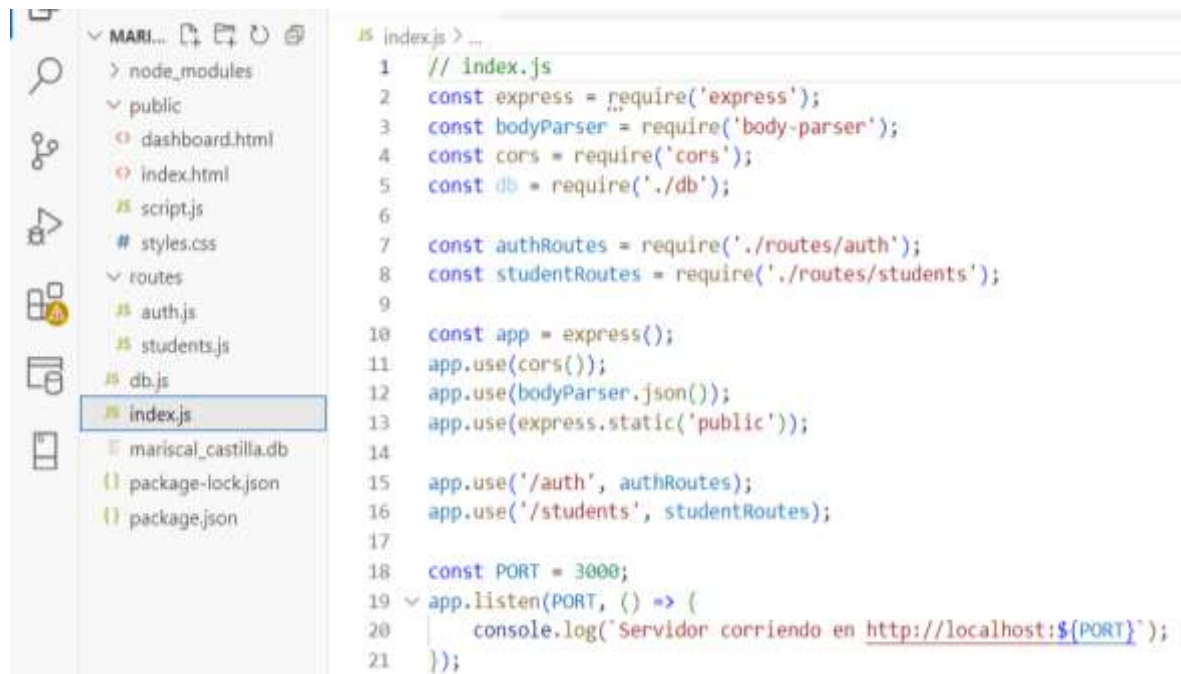




```
1 // db.js
2 const sqlite3 = require('sqlite3').verbose();
3 const db = new sqlite3.Database('mariscal_castilla.db');
4
5 db.serialize(() => {
6   // Crear tabla de usuarios
7   db.run(`
8     CREATE TABLE IF NOT EXISTS users (
9       id INTEGER PRIMARY KEY AUTOINCREMENT,
10      username TEXT UNIQUE,
11      password TEXT
12    );
13 `);
14
15   // Crear tabla de estudiantes
16   db.run(`
17     CREATE TABLE IF NOT EXISTS students (
18       id INTEGER PRIMARY KEY AUTOINCREMENT,
19       name TEXT,
20       grade TEXT,
21       section TEXT
22     );
23 `);
24 });
25
26 module.exports = db;
27
```

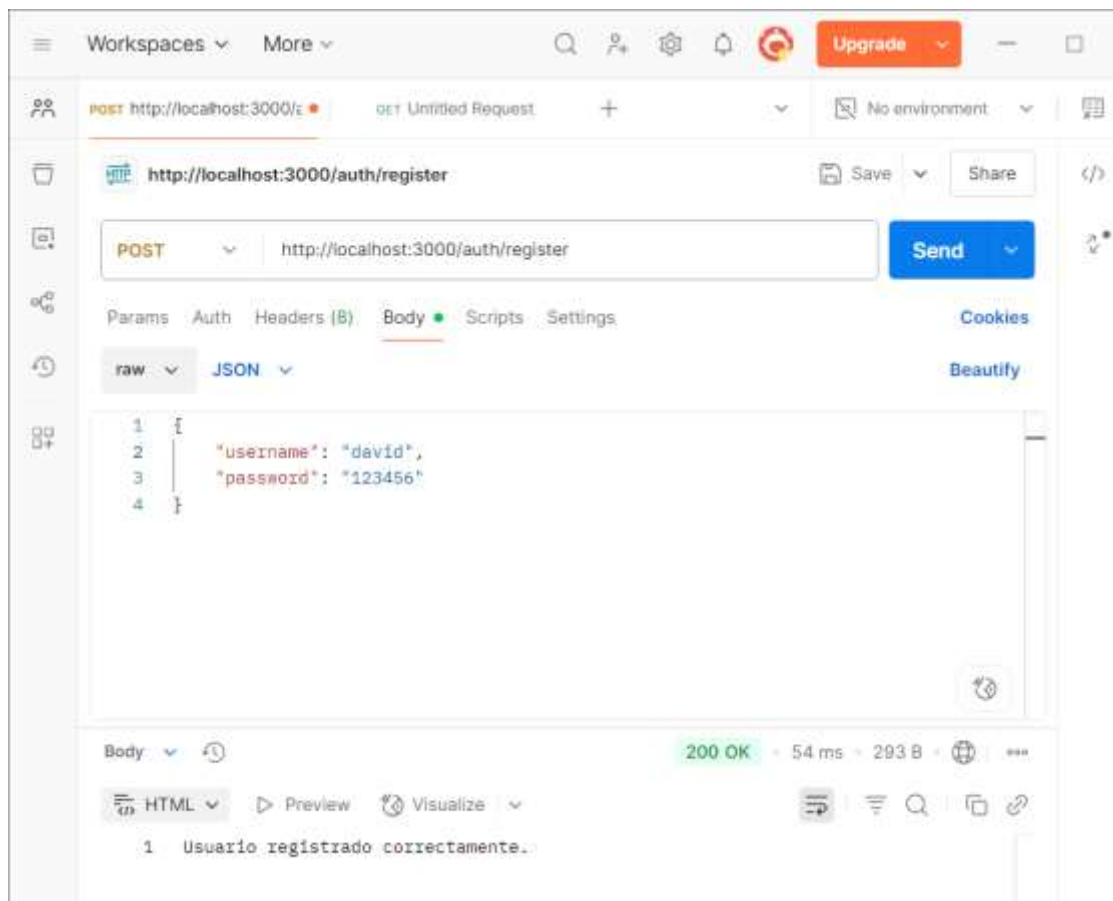
## Paso 11 – index.js

El archivo index.js configura el servidor principal de la aplicación utilizando Express.js. Se importan los módulos necesarios, como body-parser para procesar datos JSON, cors para permitir solicitudes desde otros dominios y el archivo de base de datos (db.js). Además, se incorporan las rutas de autenticación (authRoutes) y gestión de estudiantes (studentRoutes), facilitando el acceso a distintos endpoints. Finalmente, el servidor escucha en el puerto 3000, mostrando un mensaje de confirmación cuando está activo.



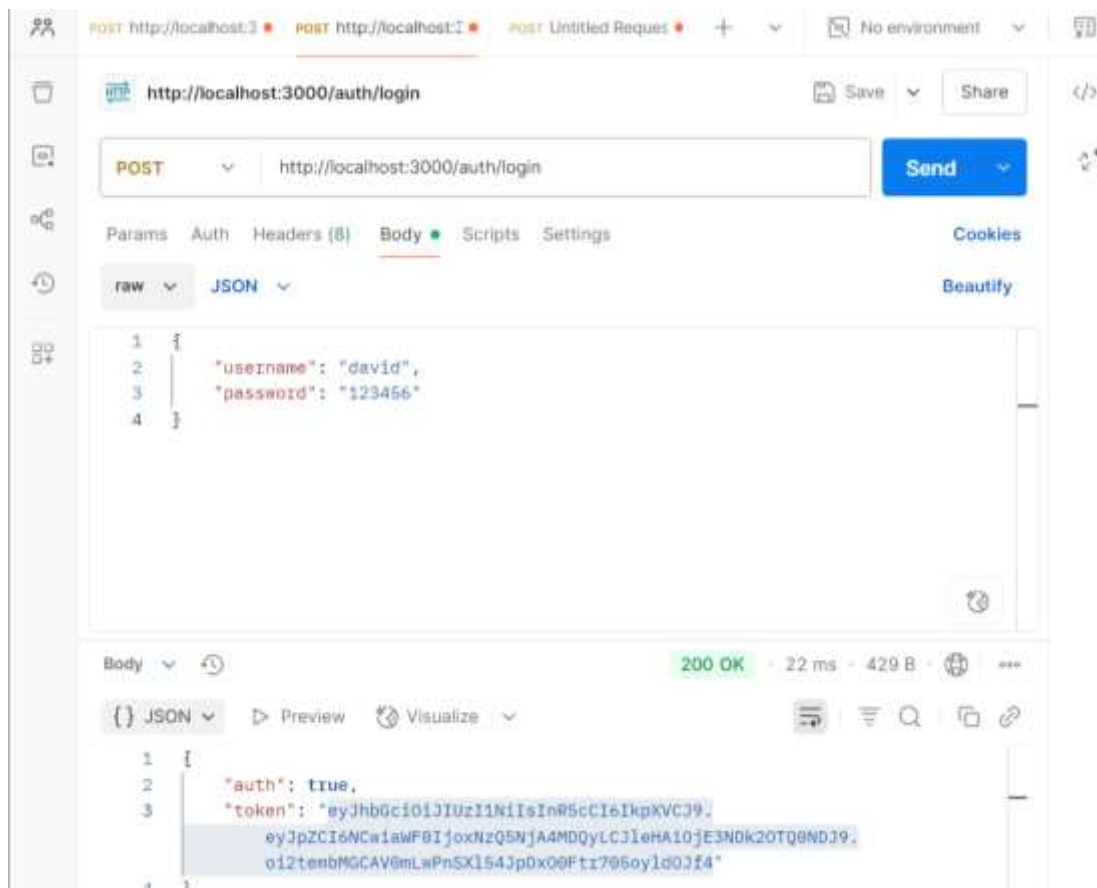
## Paso 12 – Registro en Postman

Muestra una prueba de API realizada con Postman, específicamente una solicitud POST a `http://localhost:3000/auth/register`, destinada al registro de un usuario en la aplicación. En el cuerpo de la solicitud, se envían los datos en formato JSON, incluyendo un nombre de usuario (david) y una contraseña (123456). La respuesta del servidor indica un estado 200 OK, lo que significa que el registro fue exitoso, mostrando el mensaje "Usuario registrado correctamente.". Esto confirma que el flujo de autenticación y almacenamiento de usuarios funciona correctamente dentro del sistema.



### Paso 13 – Login en Postman

Se manda una solicitud POST enviada a `http://localhost:3000/auth/login` desde Postman, con un cuerpo JSON que incluye las credenciales del usuario (username: "david" y password: "123456"). La respuesta del servidor indica una autenticación exitosa, devolviendo un objeto JSON con el campo "auth": true y un token JWT ("token": "eyJhbGciOiJIUzI1Ni..."). Este token permite el acceso seguro a recursos protegidos dentro de la aplicación. La prueba confirma que la lógica de autenticación y generación de JWT funciona correctamente.



## Paso 14 – Login en el LocalHost

El formulario de inicio de sesión para el Colegio Mariscal Castilla contiene campos para ingresar usuario y contraseña, además de un botón azul etiquetado como "Iniciar Sesión". La interfaz muestra un mensaje de bienvenida en la parte superior: "Bienvenido - Colegio Mariscal Castilla".

### Bienvenido - Colegio Mariscal Castilla

Usuario:

Contraseña:

Iniciar Sesión

## Paso 15 – Dashboard tras iniciar sesión

Se presenta una interfaz de dashboard para el registro de estudiantes. En la parte superior, hay tres campos de entrada etiquetados como "Nombre", "Grado" y "Sección", destinados a ingresar la información de un estudiante. Debajo de estos campos, se encuentra un botón azul con la etiqueta "Registrar Estudiante", que permite añadir los datos a la lista. Finalmente, en la parte inferior derecha, hay un botón azul con la etiqueta "Cerrar Sesión", que permite al usuario salir del sistema de manera segura.

### Dashboard - Registrar Estudiantes

Nombre:

Grado:

Sección:

Registrar Estudiante

Estudiantes Registrados:

Cerrar Sesión

## Paso 16 – Base de Datos DQLite

La imagen muestra la consulta de datos en la base de datos SQLite, utilizando la herramienta DB Browser for SQLite. Se ha ejecutado la consulta `SELECT * FROM students;`, mostrando una tabla con las columnas `id`, `name`, `grade` y `section`, que almacena información de los estudiantes registrados en el sistema. Los datos incluyen nombres completos y el grado correspondiente dentro de la institución Colegio Mariscal Castilla. La ejecución fue exitosa, con cinco filas retornadas en 20ms, confirmando que la base de datos está correctamente configurada y operativa.

The screenshot displays the DB Browser for SQLite application window. The title bar indicates the database file is `C:\Users\dar_f\Desktop\Mariscal-Castilla\mariscal_castilla.db`. The menu bar includes `Archivo`, `Editar`, `Ver`, `Herramientas`, and `Ayuda`. The toolbar contains icons for creating a new database, opening an existing one, saving changes, reverting changes, undoing, opening a project, saving a project, and attaching a database. The `Ejecutar SQL` button is highlighted. The `SQL1*` editor shows the following SQL query:

```
1 SELECT * FROM students;
2 SELECT * FROM users;
```

The results pane displays a table with the following data:

id	name	grade	section
1	Juan Perez	5to	A
2	LEON GARCIA DAVID DANIEL	5TO	A
3	ESPINOZA MOSALES KEYLA XIONARA	5TO	A
4	TORRES PARRGA JEFER	5TO	A
5	ROJAS SALAZAR MARTIN	5TO	A

Below the table, the execution status is shown: `Ejecución terminada sin errores. Resultado: 5 filas devueltas en 20ms. En la línea 1: SELECT * FROM students`. The right sidebar shows the `Editar celda` panel with `Modo: Texto` and the text `1 ROJAS SALAZAR MARTIN`. The `Remoto` panel shows `Identidad: Seleccione una identidad g` and `Base de datos actual`. The bottom status bar indicates `UTF-8`.

## REFERENCIAS

Google Cloud. (s.f.). *Tipos de tokens de autenticación*. Google Cloud Documentation.

Recuperado de <https://cloud.google.com/docs/authentication/token-types?hl=es>

IBM. (2025). *JSON Web Token (JWT)*. IBM Documentation. Recuperado de

<https://www.ibm.com/docs/es/cics-ts/6.x?topic=cics-json-web-token-jwt>

Microsoft. (s.f.). ¿Qué es la autenticación?. Microsoft Security. Recuperado de

<https://www.microsoft.com/es-mx/security/business/security-101/what-is-authentication#howitworks>

OpenWebinars. (s.f.). ¿Qué es *Postman*?. OpenWebinars. Recuperado el 10 de junio

de 2025, de <https://openwebinars.net/blog/que-es-postman/>