Universidad Don Bosco

CONTENIDO:

"Investigación Aplicada 2: Herramientas de Testing"

DOCENTE:

Kevin Jiménez

ASIGNATURA:

"Desarrollo de Aplicación Web con Software Interpretado en el Cliente"

INTEGRANTES:

Manuel Alejandro Chavarría Velasquez -CV211229

Daniel Adonay García Aguilar GA232128

Marvin Alexander Pérez Gómez PG240496

Vladimir Alexander López Avelar LA240481

Parte 2

Ejercicio 1: Diseñar y desarrollar una API REST para la autenticación de usuarios utilizando un framework JavaScript de su elección (por ejemplo, Express.js, Koa.js, etc.). La API deberá permitir a los usuarios registrarse, iniciar sesión, acceder a recursos protegidos y cerrar sesión de manera segura.

1. Para la creación de las Apis necesitamos tener Node.js instalado en la maquina desde la página web.

2. Luego iniciamos el proyecto con la carpeta llamada APIREST y dentro de la carpeta con visual estudio code abrimos lo que es el sistema de comando parecido a cm o PowerShell aplicando el siguiente comando (Ctrl + ñ) abrimos la pantalla de código de visual estudio code y digitamos el siguiente comando (npm init -y) y esto lo que hace es iniciar un proyecto con Node.js y dejarnos una carpeta con nombre package.json.
   Ejemplo del comando:

```
PS D:\DAW 01L\Prueba 5> npm init -y
Wrote to D:\DAW 01L\Prueba 5\package.json:

{
  "name": "prueba-5",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": ""
```

3. Después de la instalación del proyecto de Node.js instalamos de la misma manera las dependencia que vamos a ocupar con el siguiente comando (npm install express bcryptjs jsonwebtoken dotenv body-parser) lo que nos dejaría una carpeta con nombre node_modules y un archivo package-lock.json.
   Ejemplo del comando:

```
PS D:\DAW 01L\Prueba 5> npm install express bcryptjs jsonwebtoken dotenv body-parser

added 81 packages, and audited 82 packages in 3s

14 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

4. Creamos el archivo principal donde estarán alojadas las apis llamado server.js en visual estudio code :

Carga de variables de entorno: utilizara la librería dotenv para cargar variables de entrono desde un archivo .env En este caso, se utiliza para acceder a JWT_SECRET que es la clave secreta utilizada para firmar tokens JWT.

Ejemplo del código:

```
require('dotenv').config();
```

Dependencias y configuración de Express: Los siguientes comandos es para importar las librerías que ya están instaladas con los comandos anterior mente instalados con node.js.

Ejemplo del código:

```
const express = require('express');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const bodyParser = require('body-parser');
```

Luego se configura la aplicación Express para procesar JSON.

Ejemplo del código:

```
const app = express();
app.use(bodyParser.json());
```

Simulación de una base de datos: En este programa se simula una base de datos con un arreglo llamado users, que almacena los usuarios registrados. En un entorno real, este tipo de información se almacenaría en una base de datos segura.

Ejemplo del código:

```
const users = [];  // Esto simula una base de datos pero no es lo correcto.
```

Autenticación de rutas protegidas:

Este comando establece una variable llamada JWT_SECRET, que es crucial para firmar y verificar los tokens JWT (JSON Web Token) en la aplicación.

Ejemplo del código:

```
const JWT_SECRET = process.env.JWT_SECRET || 'mi_super_secreto';
```

Esta ruta permite registrar un nuevo usuario. Primero, se verifica si el nombre de usuario ya existe. Si no existe, la contraseña se cifra utilizando bcrypt antes de almacenarla. Luego, se guarda el usuario en el arreglo users y se responde con un mensaje de éxito por que se ase esto es para simular un login de la vida real y nadie pueda tomar esa información encriptándola.

Ejemplo del código:

```
// Función para autenticar y proteger rutas
function authenticateToken(req, res, next) {
    const token = req.headers['authorization']?.split(' ')[1];
    if (!token) return res.sendStatus(401);

    jwt.verify(token, JWT_SECRET, (err, user) => {
        if (err) return res.sendStatus(403);
        req.user = user;
        next();
    });
}
```

Inicio de sesión (Ruta /api/login) :

En esta ruta, el usuario proporciona su nombre de usuario y contraseña. Primero, se busca el usuario en el arreglo users. Si el usuario existe, se compara la contraseña proporcionada con la almacenada usando bcrypt.compare. Si coincide, se genera un token JWT con una duración de 1 hora y se envía como respuesta.

Ejemplo del código:

```
// 1. Registro de usuario
app.post('/api/register', async (req, res) => {
    const { username, password, email } = req.body;

    // Verifica si el usuario ya existe
    if (users.find(user => user.username === username)) {
        return res.status(400).json({ message: 'El usuario ya existe' });
    }

    // Cifra la contraseña antes de guardarla
    const hashedPassword = await bcrypt.hash(password, 10);

    const newUser = { username, email, password: hashedPassword };
    users.push(newUser);

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

Acceso a recursos protegidos (Ruta /api/protected-resource):

Esta ruta está protegida mediante la función authenticateToken. Solo los usuarios que envíen un token JWT válido pueden acceder a este recurso. Si el token es válido, se responde con un mensaje de éxito y los datos del usuario.

Ejemplo del código:

```
// 2. Inicio de sesión
app.post('/api/login', async (req, res) => {
    const { username, password } = req.body;

    const user = users.find(user => user.username === username);
    if (!user) {
        return res.status(400).json({ message: 'Usuario no encontrado' });
    }

    // Verifica la contraseña
    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
        return res.status(400).json({ message: 'Contraseña incorrecta' });
    }

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

    res.status(200).json({ message: 'Inicio de sesión exitoso', token });
});
```

Acceso a recursos protegidos (Ruta /api/protected-resource):

Esta ruta está protegida mediante la función authenticateToken. Solo los usuarios que envíen un token JWT válido pueden acceder a este recurso. Si el token es válido, se responde con un mensaje de éxito y los datos del usuario.

Ejemplo del código:

```
// 3. Recurso protegido
app.get('/api/protected-resource', authenticateToken, (req, res) => {
    res.status(200).json({ message: 'Acceso al recurso protegido', user: req.user });
});
```

Cierre de sesión (Ruta /api/logout):

Aunque JWT no requiere un mecanismo específico de cierre de sesión, en esta ruta se podría implementar una lista de tokens inválidos (lista negra). En este ejemplo, simplemente se responde con un mensaje de éxito sin implementar dicha lista.

Ejemplo del código:

```
// 4. Cierre de sesión
app.post('/api/logout', authenticateToken, (req, res) => {

    res.status(200).json({ message: 'Cierre de sesión exitoso' });
});
```

Inicio del servidor:

Aquí se inicializa el servidor Express en el puerto definido por la variable de entorno PORT o el puerto 3000 por defecto.

Ejemplo del código:

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

5. Configurar variables de entorno:
   Crea un archivo .env en la raíz de tu proyecto y define tu secreto JWT
   Ejemplo:

.env

```
#asigna una clave secreta para el uso de JWTs,
#permitiendo que tu aplicación pueda firmar y verificar los tokens para
#autenticar usuarios de manera segura.

JWT_SECRET=mi_super_secreto
```

6. Iniciar servidor:

Abrimos una terminal en visual estudio code para poder iniciar el servidor y poder visualizar las Apis y probarlas como (Introduction ,cURL,HTTPie,Curlie, etc.)

Ejemplo del comando:

7. Probar la API en Postman:

Ahora que el servidor esta en funcionamiento probaremos la APIS en postman y para que sirve Postman sirve para probar Apis sin interfaz grafica.

Caso de prueba 1: Verificar el registro de usuario

○ Descripción: Verificar que un usuario pueda registrarse

correctamente.

○ Método: POST

○ Endpoint: /api/register

○ Cuerpo de la solicitud:

{

"username": "nuevo_usuario",

"password": "password_seguro",

"email": "correo@ejemplo.com"

}

○ Verificación:

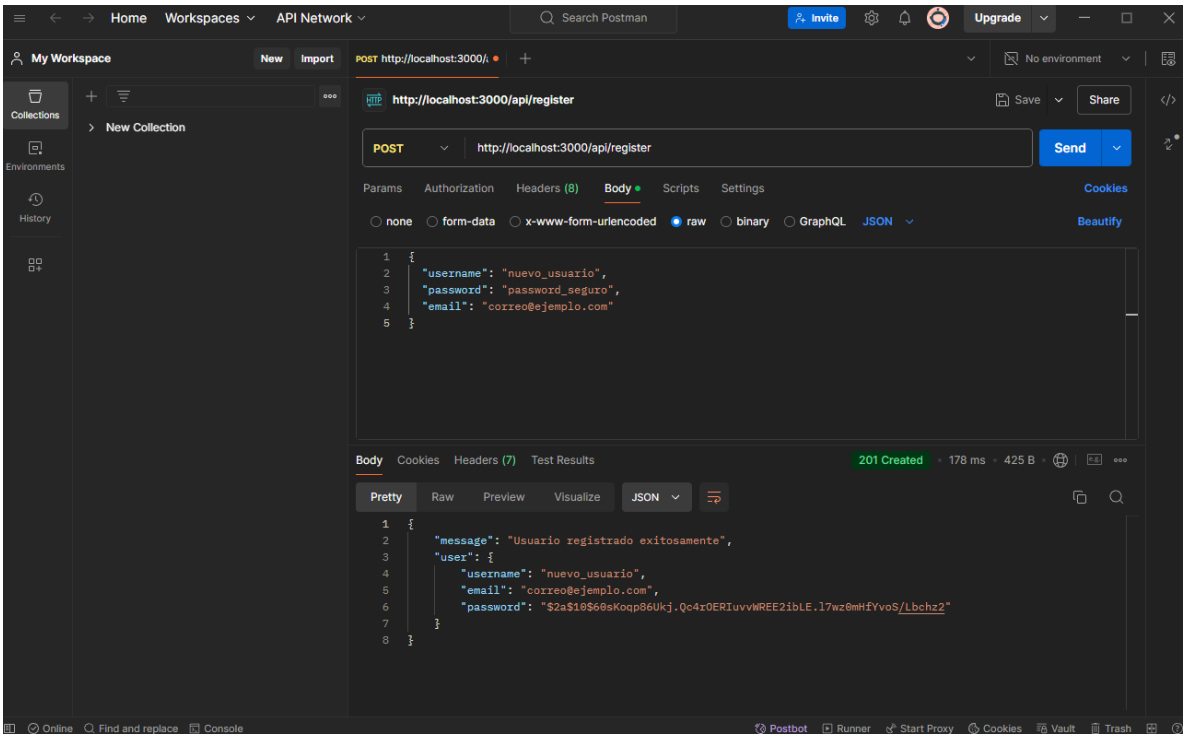
● Enviar una solicitud POST con los datos del nuevo usuario.

● Verificar que la respuesta tenga un código de estado 201

(Creado).

● Verificar que la respuesta contenga un mensaje de éxito o los datos del usuario registrado.

Imagen de la prueba realizada:



Caso de prueba 2: Verificar el inicio de sesión de usuario

○ Descripción: Verificar que un usuario pueda iniciar sesión correctamente.

○ Método: POST

○ Endpoint: /api/login

○ Cuerpo de la solicitud:

{

"username": "usuario_existente",

"password": "password_correcto"

}

○ Verificación:

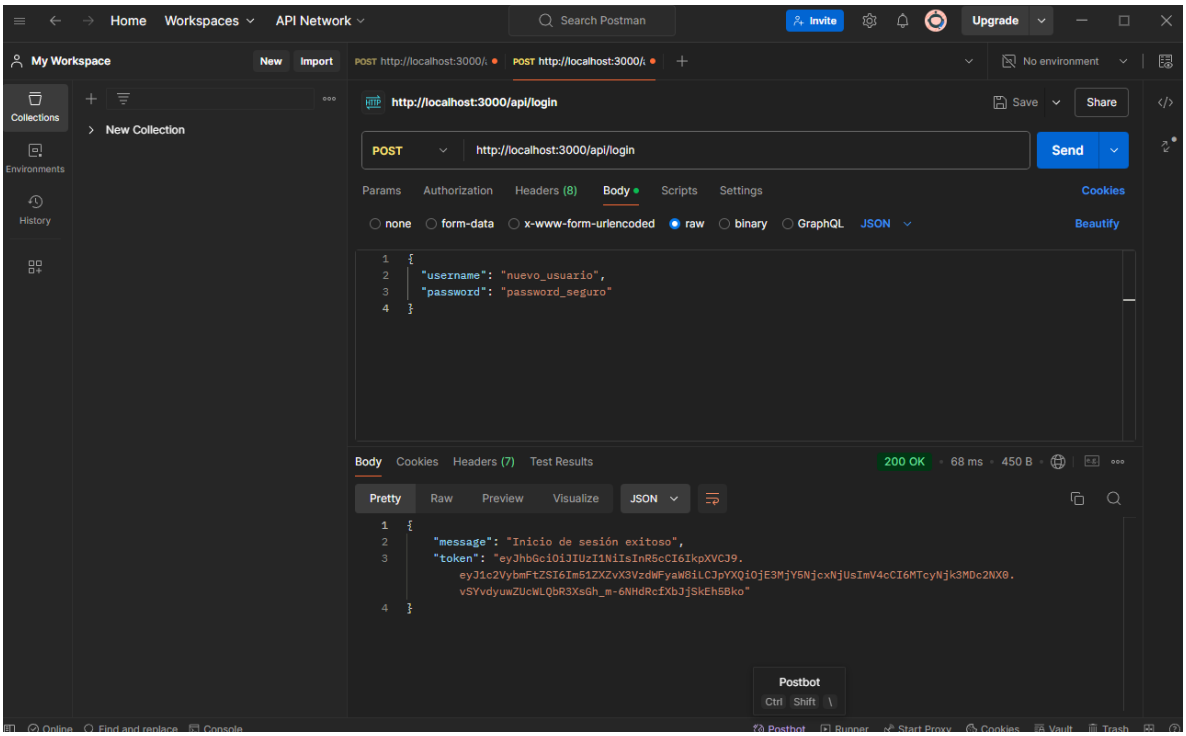
■ Enviar una solicitud POST con las credenciales del usuario.

■ Verificar que la respuesta tenga un código de estado 200 (OK).

■ Verificar que la respuesta contenga un token de

autenticación.

Imagen de la prueba realizada:



Caso de prueba 3: Verificar el acceso a un recurso protegido con autenticación

○ Descripción: Verificar que solo los usuarios autenticados puedan acceder a un recurso protegido.

○ Método: GET

○ Endpoint: /api/protected-resource

○ Encabezado de la solicitud:

{

"Authorization": "Bearer token_valido"

}

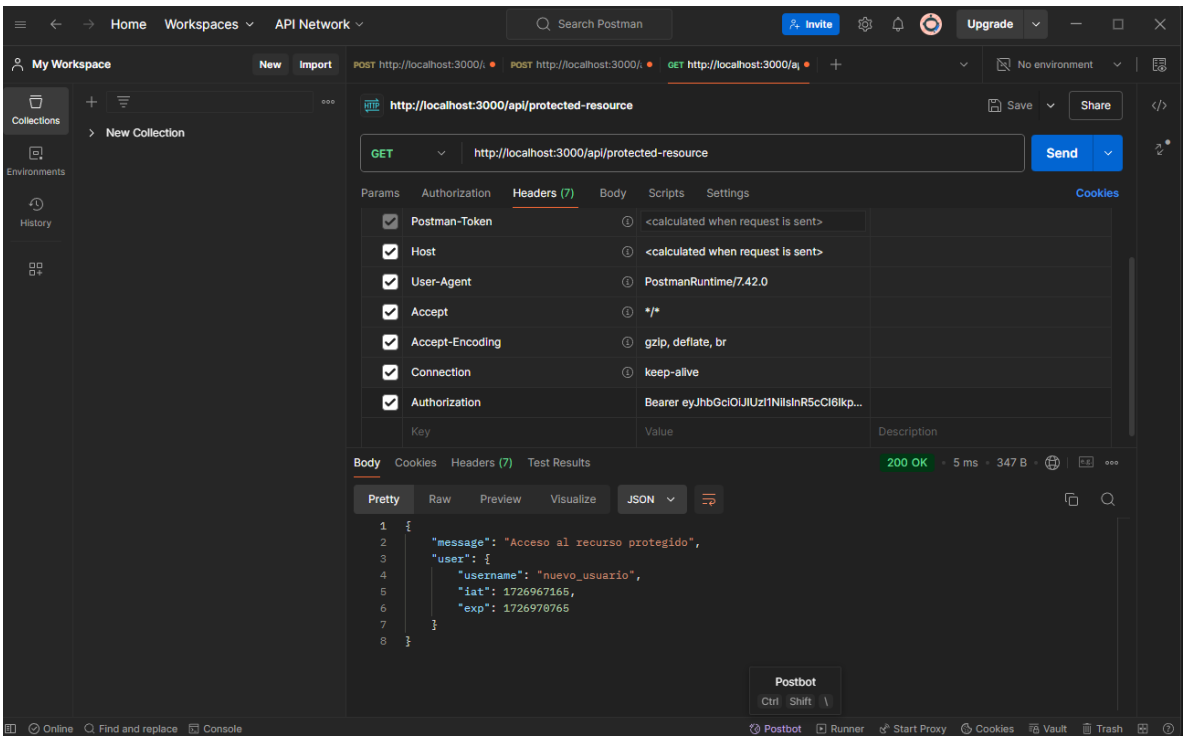
○ Verificación:

■ Enviar una solicitud GET con un token de autenticación válida.

■ Verificar que la respuesta tenga un código de estado 200 (OK).

■ Verificar que la respuesta contenga los datos del recurso protegido.

Imagen de la prueba realizada:



Caso de prueba 4: Verificar el cierre de sesión de usuario

○ Descripción: Verificar que un usuario pueda cerrar sesión correctamente.

○ Método: POST

○ Endpoint: /api/logout

○ Encabezado de la solicitud:

{

"Authorization": "Bearer token_valido"

}

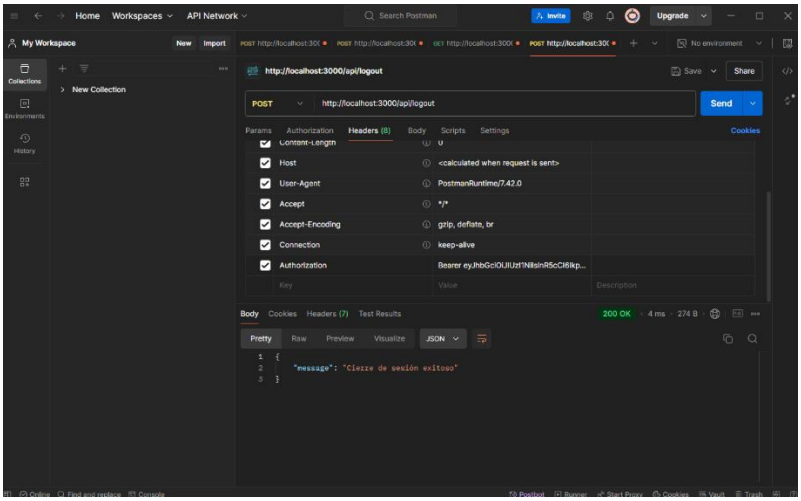
○ Verificación:

■ Enviar una solicitud POST con el token de autenticación.

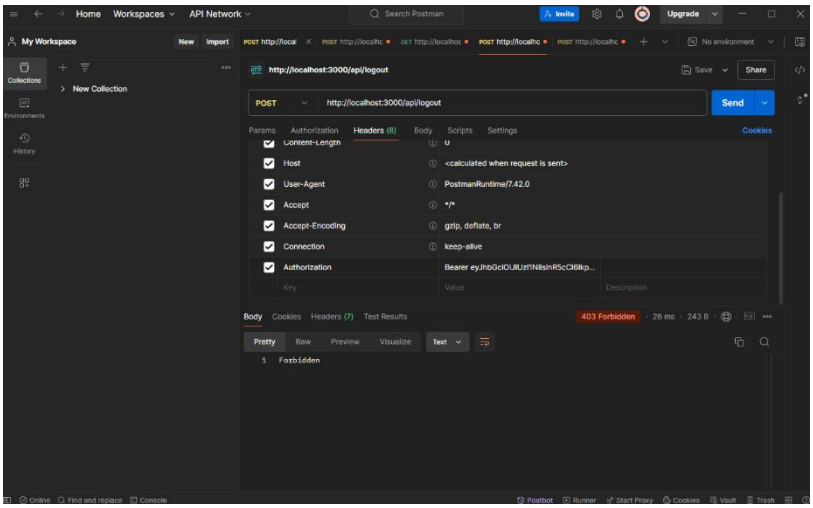
■ Verificar que la respuesta tenga un código de estado 200 (OK).

■ Verificar que el token de autenticación ya no sea válido para futuras solicitudes.

Imagen de la prueba realizada:



Esta imagen es la verificación de autenticación que ya no es válida para las solicitud.



Bibliografía

[1]Registro y Login funcional en Javascript con node y express - guía paso a paso.

Disponible en: https://www.youtube.com/watch?v=fjuIeiA8QkE&t=266s

[2]"Postman en Español: ¡El tutorial definitivo para el éxito en las pruebas API!.

Disponible en:

https://www.youtube.com/watch?v=qsejysrhJiU&t=108s

[3] API REST con Node js y Express | CRUD

Disponible en:

https://www.youtube.com/watch?v=BImKbdy-ubM&t=28s

[4] Cómo CONSUMIR una API REST con JAVASCRIPT y Fetch + Promises con gestión de Errores

Disponible en:

https://www.youtube.com/watch?v=FJ-w0tf3d_w