

Universidad Católica Boliviana “San Pablo”



Segunda Evaluación - Tecnologías Web II (1-2025)

Microservicio WebSocket para la Creación de Documentos en el Sistema
Repositorio Documentos

Autor: Marco Octavio Luna Vargas

Materia: Tecnologías Web II

Docente: Miguel Angel Pacheco Arteaga

Fecha: 4 de mayo de 2025

1. Análisis del Proyecto Integrador

- Identificación clara del endpoint

Endpoint consumido:

POST /api/documentos/crear

Autenticación previa:

POST /api/usuarios/login

Se envía el correo y contraseña y se recibe un token en el campo token, que es requerido para consumir el endpoint de creación de documentos.

- Justificación de la elección del endpoint

Se eligió el endpoint */api/documentos/crear* porque permite integrar funcionalidades reales del sistema, como la creación de documentos históricos con datos completos. Además, su implementación es directa y está bien estructurada, ideal para consumir desde un microservicio.

- Descripción técnica del endpoint

Característica	Valor
Método	POST
URL	/api/documentos/crear
Autenticación	Token JWT en Authorization
Formato de envío	JSON en el body
Campos requeridos	titulo, autor, fecha, tipo, categoria, archivo_url, descripcion, usuario_responsable

2. Diseño del Microservicio

- Objetivo del Microservicio

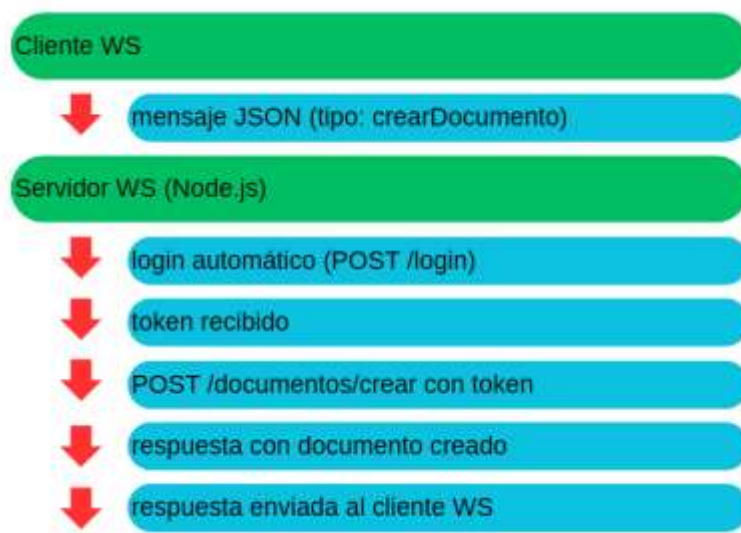
Construir un microservicio en Node.js que utilice WebSockets para recibir peticiones en tiempo real, autenticarse automáticamente y crear documentos en el sistema principal de forma asíncrona.

- Tecnología usada: WebSockets

Se eligió WebSockets porque:

- Permite comunicación en tiempo real.
- Evita la sobrecarga de peticiones HTTP repetidas.
- Ideal para escenarios donde se requiere respuesta inmediata del backend al cliente.

- Diagrama de Flujo de Integración



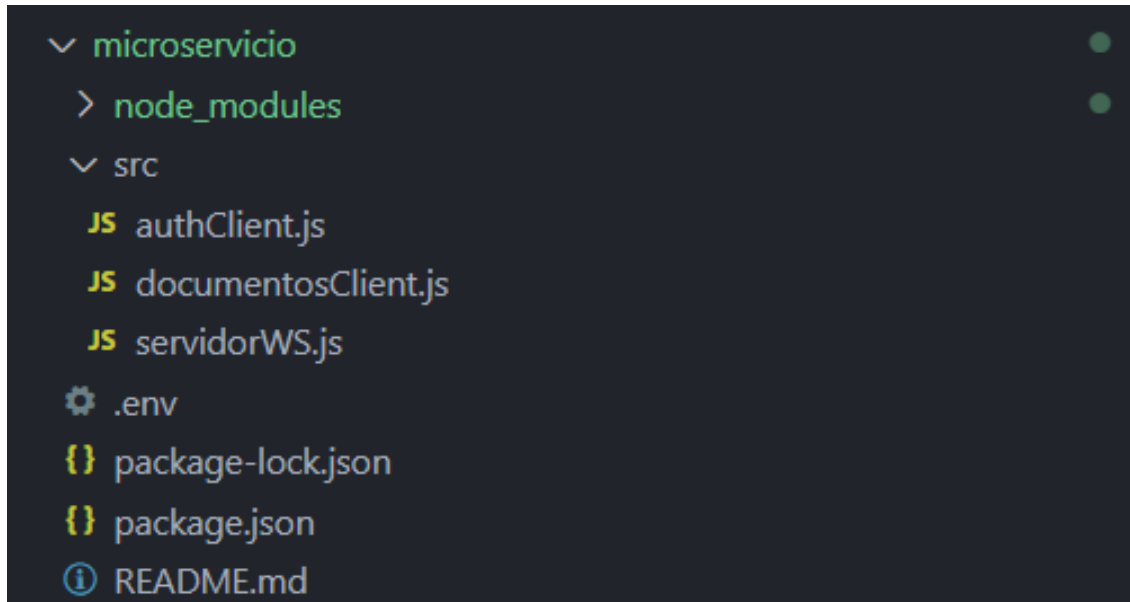
3. Implementación Técnica

- Tecnologías usadas

El microservicio fue desarrollado utilizando **Node.js** como entorno de ejecución y el protocolo **WebSocket** mediante la biblioteca **ws** para la comunicación en tiempo real. Se utilizó **Axios** para realizar las peticiones HTTP al backend REST externo y **dotenv** para gestionar las variables de entorno de forma segura. Toda la lógica está escrita en **JavaScript (ES6)**, y el backend al que se conecta utiliza **MongoDB** como base de datos. Para las pruebas de funcionamiento se emplearon herramientas como **Postman** para validar la

autenticación y **WebSocket King** para enviar mensajes al microservicio y verificar las respuestas en tiempo real.

- Estructura del Proyecto



- Código base

.env

```
1  API_URL=http://localhost:5000
2  CORREO=marco.luna.v@ucb.edu.bo
3  CONTRASENA=admin
4
```

authClient.js

```
1 // src/authClient.js
2 const axios = require('axios');
3
4 // src/documentosClient.js
5 const { getToken } = require('./authClient');
6 require('dotenv').config();
7
8 const API_URL = process.env.API_URL;
9
10 async function crearDocumento(datosDocumento) {
11   try {
12     const token = await getToken();
13
14     const response = await axios.post(`${API_URL}/api/documentos/crear`, datosDocumento, {
15       headers: {
16         Authorization: `Bearer ${token}`,
17       },
18     });
19
20     return response.data;
21   } catch (error) {
22     console.error('❌ Error al crear el documento:', error.response?.data || error.message);
23     throw new Error('No se pudo crear el documento');
24   }
25 }
26
27 module.exports = { crearDocumento };
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

documentosClient.js

```
1 // src/documentosClient.js
2 const axios = require('axios');
3 const { getToken } = require('./authClient');
4 require('dotenv').config();
5
6 const API_URL = process.env.API_URL;
7
8 async function crearDocumento(datosDocumento) {
9   try {
10     const token = await getToken();
11
12     const response = await axios.post(`${API_URL}/api/documentos/crear`, datosDocumento, {
13       headers: {
14         Authorization: `Bearer ${token}`,
15       },
16     });
17
18     return response.data;
19   } catch (error) {
20     console.error('❌ Error al crear el documento:', error.response?.data || error.message);
21     throw new Error('No se pudo crear el documento');
22   }
23 }
24
25 module.exports = { crearDocumento };
26
```

servidorWS.js

```
1 // src/servidorWS.js
2 const WebSocket = require('ws');
3 const { crearDocumento } = require('./documentosClient');
4
5 const wss = new WebSocket.Server({ port: 4001 }, () => {
6   console.log('✅ Servidor WebSocket escuchando en ws://localhost:4001');
7 });
8
9 wss.on('connection', (ws) => {
10   console.log('👤 Cliente conectado');
11
12   ws.on('message', async (message) => {
13     try {
14       const data = JSON.parse(message.toString());
15
16       if (data.tipo === 'crearDocumento' && data.documento) {
17         console.log('📄 Recibido documento:', data.documento);
18
19         const resultado = await crearDocumento(data.documento);
20
21         ws.send(JSON.stringify({
22           tipo: 'respuesta',
23           estado: 'ok',
24           mensaje: 'Documento creado con éxito',
25           datos: resultado,
26         }));
27       } else {
28         ws.send(JSON.stringify({
29           tipo: 'error',
30           mensaje: 'Formato no válido. Se esperaba tipo="crearDocumento" y un objeto documento.',
31         }));
32       }
33     } catch (error) {
34       ws.send(JSON.stringify({
35         tipo: 'error',
36         mensaje: 'Error procesando solicitud: ' + error.message,
37       }));
38     }
39   });
40
41   ws.on('close', () => {
42     console.log('👤 Cliente desconectado');
43   });
44 });
45
```

Librerías necesarias: npm install ws axios dotenv

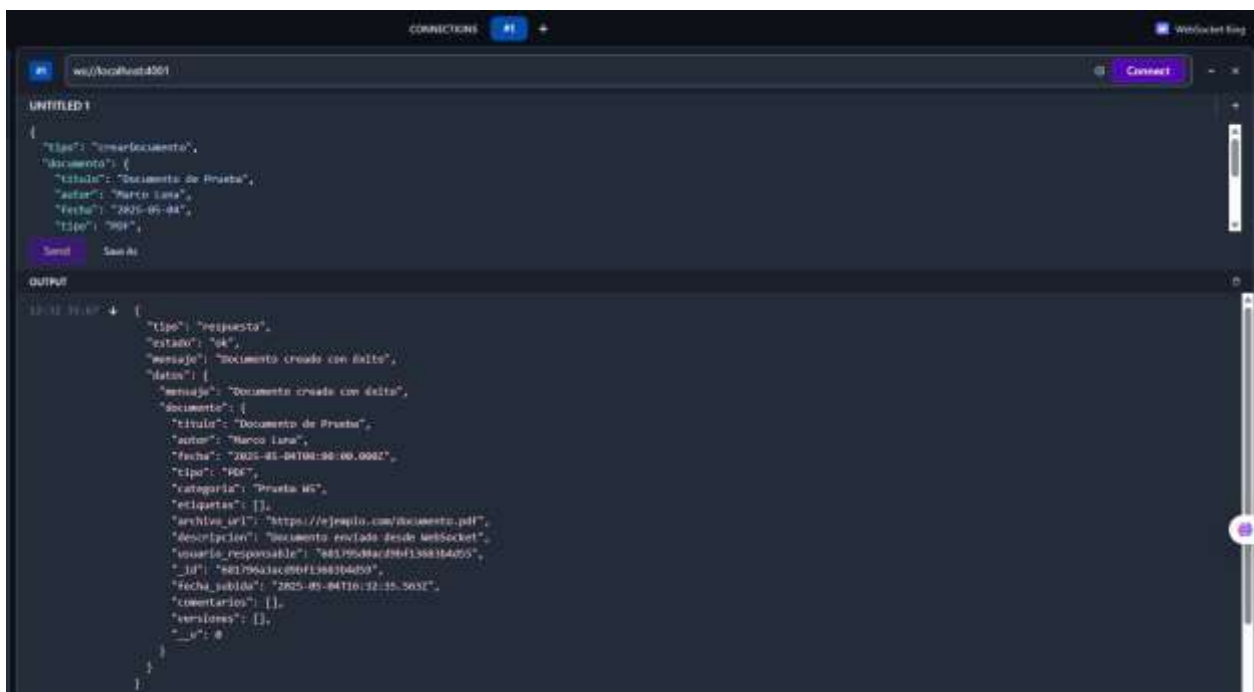
4. Pruebas y Documentación

- Evidencias funcionales

Ejecución del microservicio en consola. Se muestra que el servidor WebSocket está escuchando correctamente en `ws://localhost:4001`, que el cliente se conectó exitosamente y que se recibió un documento con sus respectivos datos: título, autor, fecha, tipo, categoría, URL del archivo, descripción y el ID del usuario responsable.

```
PS D:\Octavio-principal\Dropbox\Desk\University\tecweb2\evaluacion\microservicio> node src/servidorWS.js
✓ Servidor WebSocket escuchando en ws://localhost:4001
🔗 Cliente conectado
📄 Recibido documento: {
  titulo: 'Documento de Prueba',
  autor: 'Marco Luna',
  fecha: '2025-05-04',
  tipo: 'PDF',
  categoria: 'Prueba WS',
  archivo_url: 'https://ejemplo.com/documento.pdf',
  descripcion: 'Documento enviado desde WebSocket',
  usuario_responsable: '681795d8acd9bf13683b4d55'
}
```

Se muestra el microservicio WebSocket en funcionamiento dentro de WebSocket King. En la parte superior, el cliente se conecta correctamente a `ws://localhost:4001` y emite un mensaje con el tipo `crearDocumento` y los datos del nuevo documento. En la sección `OUTPUT`, se recibe una respuesta del servidor confirmando que el documento fue creado con éxito, junto con todos los campos del documento creado, incluidos el ID generado, fecha de subida y usuario responsable.



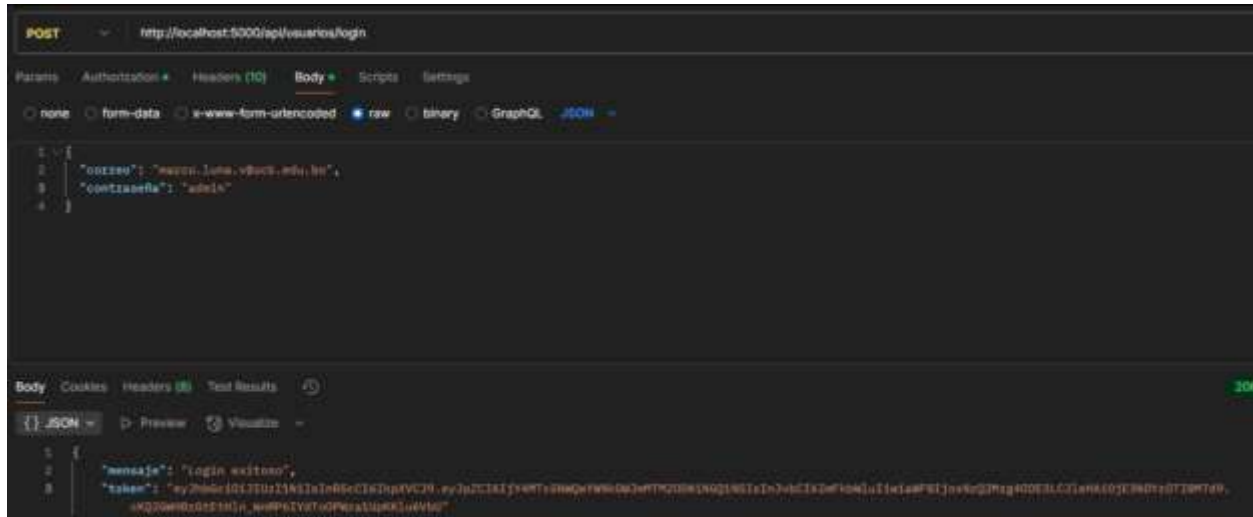
```
CONNECTIONS 1
ws://localhost:4001 [Connect]

UNTITLED 1
{
  "tipo": "crearDocumento",
  "documento": {
    "titulo": "Documento de Prueba",
    "autor": "Marco Luna",
    "fecha": "2025-05-04",
    "tipo": "PDF"
  }
}

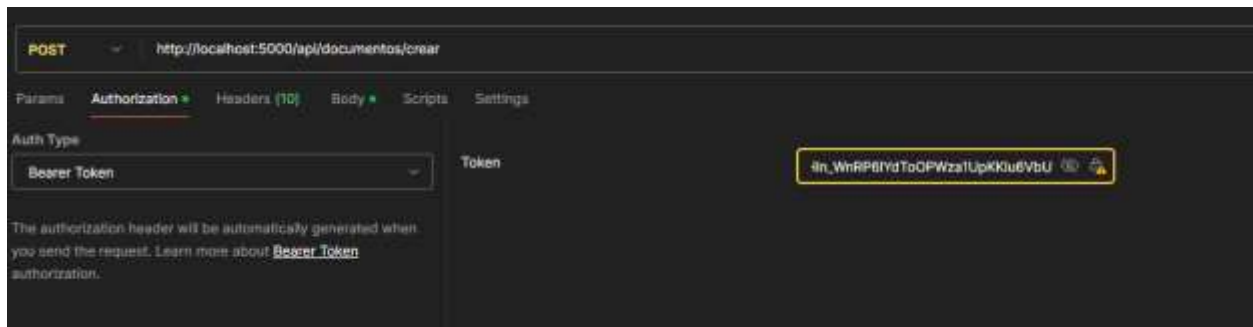
OUTPUT
12:12 PM + {
  "tipo": "respuesta",
  "estado": "OK",
  "mensaje": "Documento creado con éxito",
  "datos": {
    "mensaje": "Documento creado con éxito",
    "documento": {
      "titulo": "Documento de Prueba",
      "autor": "Marco Luna",
      "fecha": "2025-05-04T10:30:00.000Z",
      "tipo": "PDF",
      "categoria": "Prueba WS",
      "etiquetas": [],
      "archivo_url": "https://ejemplo.com/documento.pdf",
      "descripcion": "Documento enviado desde WebSocket",
      "usuario_responsable": "681795d8acd9bf13683b4d55",
      "_id": "6817963ac896f13683b4d59",
      "fecha_subida": "2025-05-04T10:32:15.503Z",
      "comentarios": [],
      "versiones": [],
      "__v": 0
    }
  }
}
```

- Pruebas manuales

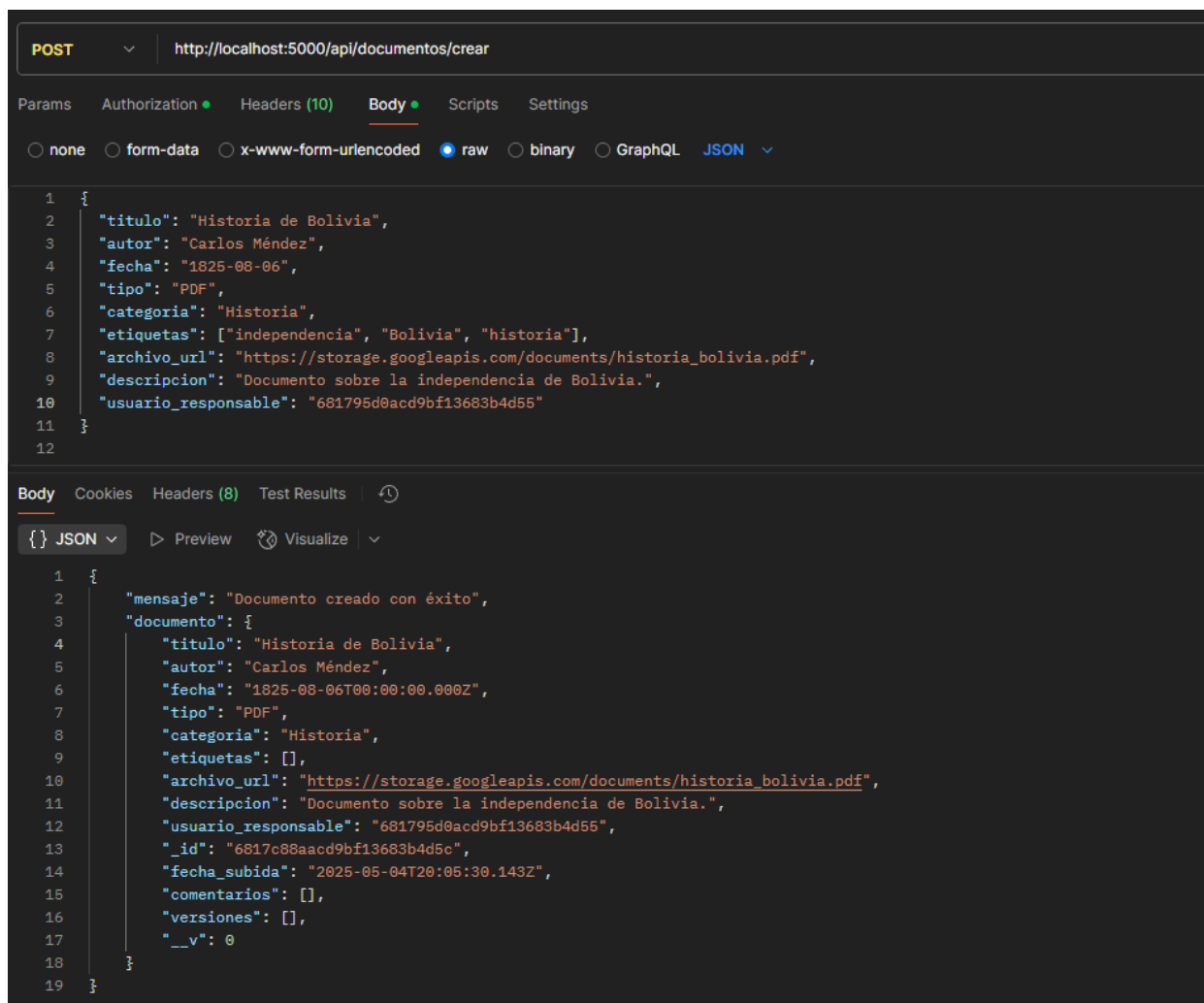
Se muestra una prueba manual del endpoint de login (`/api/usuarios/login`) realizada desde Postman. Se envían las credenciales del usuario administrador en formato JSON. La respuesta incluye un mensaje de "Login exitoso" y un token JWT, el cual es necesario para autenticarse y poder realizar la creación de documentos posteriormente.



Esta evidencia demuestra cómo se utiliza el token JWT obtenido previamente para autenticar la solicitud. Se realiza una petición POST al endpoint `/api/documentos/crear`, incluyendo el token en el encabezado `Authorization` como `Bearer Token`. Esta autenticación permite que el backend acepte la solicitud de creación de documentos.



Se observa una solicitud completa de creación de documento con todos los campos requeridos, como *título*, *autor*, *fecha*, *tipo*, *categoría*, *archivo_url*, y *descripcion*. La respuesta del backend indica que el documento fue creado correctamente, mostrando todos los datos insertados y generando un *_id* único, lo cual valida que el microservicio funciona de forma exitosa e integrada.



5. Documentación Técnica (README)

Todo el proyecto está completamente documentado en el archivo *README.md*, el cual se encuentra en la raíz del repositorio. Este archivo incluye:

- **Descripción del Proyecto:** Explicación clara del propósito del microservicio, el cual permite la creación remota de documentos mediante WebSockets y la conexión con un backend REST autenticado vía JWT.
- **Estructura de Carpetas:** Se muestra la organización del proyecto, incluyendo los archivos clave (*servidorWS.js*, *authClient.js*, *documentosClient.js*, *.env*, etc.).

- **Funcionamiento General:** Detalla el flujo completo del microservicio, desde que recibe un mensaje WebSocket hasta que realiza el login y ejecuta una petición POST al backend.
- **Autenticación:** Se explica cómo se realiza el login automático usando credenciales definidas en el archivo `.env`, junto con el formato JSON esperado.
- **Manejo de Documentos:** Se documenta la estructura del objeto `documento` que se debe enviar para su correcta creación, con todos los campos requeridos.
- **Ejemplo de Flujo Completo:** Se incluye un caso de uso completo desde la conexión del cliente WebSocket hasta la respuesta del backend, con ejemplos reales de JSON.
- **Requisitos e Instalación:** Se listan las dependencias necesarias (como Node.js), instrucciones para clonar el repositorio, instalar dependencias y ejecutar el servidor.
- **Pruebas Manuales:** Se recomienda verificar el correcto funcionamiento del backend antes de levantar el microservicio, usando herramientas como Postman.
- **Notas Finales:** Se aclara que este gateway actúa como un intermediario en tiempo real, y se enfatiza la importancia de tener el backend previamente en ejecución.

6. Conclusión

El microservicio cumple correctamente con los requerimientos del proyecto integrador, permitiendo la integración en tiempo real con el backend. La implementación en WebSocket resultó adecuada y robusta. Además, se demostró el uso real de autenticación segura y consumo de endpoints protegidos.