Práctica: Consumo de API REST con Fetch y JWT

Objetivo. Crear una aplicación web que consuma una API REST pública, maneje autenticación JWT y muestre datos dinámicos.

Herramientas Necesarias

- Navegador moderno (Chrome/Firefox).
- Editor de código (VS Code).
- API de prueba: JSONPlaceholder (posts) + ReqRes.in (autenticación JWT).

Paso 1: Configuración Inicial

1. Crea una carpeta api-practice con dos archivos:
    - index.html (estructura HTML básica).
    - app.js (script principal).

Código HTML (index.html):

```
<!DOCTYPE html>
<html>
<head>
    <title>Consumo de API REST</title>
</head>
<body>
    <h1>Posts</h1>
    <div id="posts"></div>
    <script src="app.js"></script>
</body>
</html>
```

Paso 2: Consumo Básico de API (GET)

Objetivo: Obtener y mostrar datos de JSONPlaceholder (posts).

Código JS (app.js):

```
// 1. Función para obtener posts
async function fetchPosts() {
    try {
        const response = await fetch('https://jsonplaceholder.typicode.com/posts');
        if (!response.ok) throw new Error(`HTTP error! status: ${response.status}`);
        const posts = await response.json();
        displayPosts(posts.slice(0, 5)); // Mostrar solo 5 posts
    } catch (error) {
        console.error('Error:', error);
    }
}

// 2. Función para mostrar posts en el DOM
```

```
function displayPosts(posts) {
    const container = document.getElementById('posts');
    container.innerHTML = posts
        .map(post => `<div><h3>${post.title}</h3><p>${post.body}</p></div>`)
        .join('');
}

// 3. Ejecutar al cargar la página
fetchPosts();
```

Resultado Esperado: Ver 5 posts renderizados en la página al abrir index.html en el navegador.

Paso 3: Autenticación JWT (POST)

Objetivo: Simular login y obtener un token JWT usando ReqRes.in.

Código JS (agregar a app.js):

```
// 4. Función de login
async function loginUser(email, password) {
    try {
        const response = await fetch('https://reqres.in/api/login', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ email, password }),
        });
        if (!response.ok) throw new Error(`Error: ${response.status}`);
        const data = await response.json();
        return data.token; // Retorna el token JWT
    } catch (error) {
        console.error('Login fallido:', error);
    }
}

// 5. Ejemplo de uso (simular login)
async function main() {
    const token = await loginUser('eve.holt@reqres.in', 'cityslicka'); // Credenciales de
prueba
    if (token) {
        console.log('Token JWT:', token);
        // ¡Ahora puedes usar este token en otras peticiones!
    }
}

main();
```

Prueba:

- Abre la consola del navegador (F12).
- Deberías ver el token JWT impreso.

Paso 4: Integración con Autenticación (GET Autorizado)

Objetivo: Usar el token JWT para acceder a un recurso protegido (simulado).

Código JS (agregar a app.js):

```
// 6. Función para obtener datos protegidos con JWT
async function fetchProtectedData(token) {
    try {
        const response = await fetch('https://jsonplaceholder.typicode.com/users/1', {
            headers: { 'Authorization': `Bearer ${token}` }, // Simulación
        });
        if (!response.ok) throw new Error(`Error: ${response.status}`);
        const user = await response.json();
        console.log('Usuario protegido:', user);
    } catch (error) {
        console.error('Error al obtener datos protegidos:', error);
    }
}

// Modificar la función main para incluir esto:
async function main() {
    const token = await loginUser('eve.holt@reqres.in', 'cityslicka');
    if (token) {
        await fetchProtectedData(token); // Usar el token
    }
}
```

Explicación:

- Aunque JSONPlaceholder no requiere autenticación, simulamos el flujo añadiendo el header Authorization.

Desafío para el Estudiante (Tarea) - Manejo de Errores y Validación

1. Modifica fetchPosts para mostrar un mensaje de error en el DOM si la API falla.
2. Valida que el email y password no estén vacíos antes de hacer login.
3. **Bonus:** Implementa un formulario HTML para ingresar email y password dinámicamente.

Recursos Adicionales:

- Documentación Fetch API.
- JWT Debugger para inspeccionar tokens.
- Ejemplo en CodeSandbox.

Evaluación:

- **Criterios:**
    - Correcto uso de fetch y manejo de errores.

    - Integración exitosa del token JWT en headers.
    - Calidad del código (legibilidad, comentarios).
- **Entrega:** Publicar el código en GitHub y compartir el enlace.

Con esta práctica, los estudiantes aplican conceptos clave (peticiones HTTP, autenticación, DOM) en un escenario realista.

Práctica: Autenticación JWT en Peticiones HTTP

Objetivo. Crear una aplicación web que genere tokens JWT manualmente, los valide y los use en peticiones GET/POST protegidas.

Herramientas Necesarias

- Navegador moderno (Chrome/Firefox).
- Editor de código (VS Code).
- JWT.io: Para generar/validar tokens.
- API de prueba: JSONPlaceholder (simulación de endpoints protegidos).

Paso 1: Configuración Inicial

1. Crea una carpeta jwt-practice con:
    - index.html (estructura HTML + formulario).
    - app.js (lógica de autenticación y peticiones).

Código HTML (index.html):

```
html
Copy
<!DOCTYPE html>
<html>
<head>
    <title>Práctica JWT</title>
</head>
<body>
    <h1>Autenticación con JWT</h1>
    <button onclick="fetchDataWithJWT()">Obtener Datos Protegidos (GET)</button>
    <form id="postForm">
        <input type="text" id="data" placeholder="Ingresa datos" required>
        <button type="submit">Enviar Datos (POST)</button>
    </form>
    <div id="response"></div>
    <script src="app.js"></script>
</body>
</html>
```

Paso 2: Generar un Token JWT Manualmente

Objetivo: Crear un token JWT válido usando JWT.io para simular autenticación.

1. **Genera el token:**
    - Ve a JWT.io.
    - En el **Payload**, ingresa:

```
{
  "userId": 1,
  "username": "usuario_prueba",
  "exp": 1735689600  // Fecha de expiración (1 Enero 2025)
}
```

- Usa el **secreto**: clave_secreta_educativa (simulación para fines didácticos).
- Copia el token generado (ej: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjEsInVzZXJuYW1lIjoidXN1YXJpb19wcnVlYmEiLCJleHAiOjE3MzU2ODk2MDB9.4V7nO7n0Jf8Z8XvOJ8Z8XvOJ8Z8XvOJ8Z8XvOJ8Z8XvO).

Paso 3: Implementar Lógica JWT en JavaScript

Código JS (app.js):

```
// 1. Token JWT generado manualmente (simulación)
const JWT_TOKEN = 'TU_TOKEN_GENERADO_EN_JWT_IO'; // Reemplaza con tu token

// 2. Función para validar el token (simulación)
function validateToken(token) {
    try {
        // Decodificar el token (sin verificar firma)
        const payload = JSON.parse(atob(token.split('.')[1]));
        if (payload.exp < Date.now() / 1000) throw new Error("Token expirado");
        return true;
    } catch (error) {
        console.error("Token inválido:", error);
        return false;
    }
}

// 3. Función GET con JWT
async function fetchDataWithJWT() {
    if (!validateToken(JWT_TOKEN)) {
        alert("Token inválido o expirado");
        return;
    }

    try {
        const response = await fetch('https://jsonplaceholder.typicode.com/posts/1', {
            headers: { 'Authorization': `Bearer ${JWT_TOKEN}` }
        });
        const data = await response.json();
        document.getElementById('response').innerHTML = `<pre>${JSON.stringify(data, null,
2)}</pre>`;
    } catch (error) {
        console.error("Error GET:", error);
    }
}

// 4. Función POST con JWT
document.getElementById('postForm').addEventListener('submit', async (e) => {
    e.preventDefault();
    if (!validateToken(JWT_TOKEN)) {
        alert("Token inválido o expirado");
        return;
```

```
    }

    const postData = {
        title: document.getElementById('data').value,
        body: 'Contenido de prueba',
        userId: 1
    };

    try {
        const response = await fetch('https://jsonplaceholder.typicode.com/posts', {
            method: 'POST',
            headers: {
                'Authorization': `Bearer ${JWT_TOKEN}`,
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(postData)
        });
        const data = await response.json();
        document.getElementById('response').innerHTML = `<pre>${JSON.stringify(data, null,
2)}</pre>`;
    } catch (error) {
        console.error("Error POST:", error);
    }
});
```

Paso 4: Probar la Aplicación

1. **GET Protegido:**
    - Haz clic en el botón "Obtener Datos Protegidos".
    - Verifica que el token se envía en el header Authorization.
    - Deberías ver el post con ID 1 en pantalla.
2. **POST Protegido:**
    - Ingresa texto en el formulario y envía.
    - La API retornará un objeto simulado con ID 101.

Desafíos para el Estudiante (tarea)

1. **Implementar Expiración Real:**
    - Modifica el payload en JWT.io con exp: Math.floor(Date.now() / 1000) + 60 (1 minuto).
    - Agrega un mensaje claro cuando el token expire.
2. **Almacenamiento Seguro:**
    - Usa localStorage para guardar el token en lugar de una variable global.
3. **Validación Avanzada:**

- Usa jsonwebtoken en un backend Node.js para validar la firma del token (ejercicio adicional).

Recursos Adicionales

- JWT Debugger: Para inspeccionar tokens.
- Documentación Fetch.
- Ejemplo en CodeSandbox.

Evaluación

- **Criterios:**
  - Integración correcta del token en headers.
  - Manejo de errores (token expirado, respuestas HTTP no exitosas).
  - Calidad del código (uso de async/await, legibilidad).
- **Entrega:** Publicar el código en GitHub y compartir capturas de las peticiones en la consola del navegador.

Con esta práctica, los estudiantes entenderán el flujo completo de autenticación JWT en el cliente, desde la generación hasta su uso en peticiones reales.