JSONPlaceholder

1. ¿Qué es JSONPlaceholder?

JSONPlaceholder es un servicio web gratuito y público que ofrece varias rutas (**endpoints**) para trabajar con **datos de prueba en formato JSON**. Está diseñado para realizar experimentos y pruebas de funcionalidad sin necesidad de configurar un servidor propio. Los endpoints simulan los métodos más comunes de una API REST, como:

- **GET** (obtener datos),
- **POST** (enviar datos),
- **PUT** (actualizar datos),
- **PATCH** (modificar parcialmente datos),
- **DELETE** (eliminar datos).

Con JSONPlaceholder, puedes practicar cómo realizar consultas asíncronas a un servidor y cómo manejar datos en formato JSON sin correr el riesgo de romper una base de datos real.

2. ¿Qué contiene?

JSONPlaceholder provee varios conjuntos de datos (endpoints) que simulan recursos típicos de una aplicación. Algunos de los más utilizados son:

1. **/posts**: lista de publicaciones (posts) ficticias.
2. **/comments**: lista de comentarios de ejemplo.
3. **/albums**: lista de álbumes.
4. **/photos**: lista de fotos.
5. **/todos**: lista de tareas o pendientes.
6. **/users**: lista de usuarios de ejemplo.

Cada endpoint devuelve un listado de objetos JSON. Por ejemplo, al hacer una petición GET a https://jsonplaceholder.typicode.com/posts, recibirás un array de objetos con información de publicaciones (id, userId, title, body).

3. Dos ejemplos de uso básico para descarga de datos (GET) con JavaScript y HTML

Para estos ejemplos, utilizaremos el método **fetch()** de JavaScript, que permite realizar peticiones asíncronas a un servidor y manejar la respuesta.

Ejemplo 1: Listar publicaciones en consola

HTML (index.html):

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <title>Descarga de datos - Ejemplo 1</title>
</head>
<body>

  <h1>Descargar Publicaciones (JSONPlaceholder)</h1>
  <button id="btnDescargar">Descargar Posts</button>

  <script>
    document.getElementById('btnDescargar').addEventListener('click', () => {
      fetch('https://jsonplaceholder.typicode.com/posts')
        .then(response => response.json())
        .then(data => {
          console.log("Datos recibidos:", data);
        })
        .catch(error => {
          console.error("Error al descargar:", error);
        });
    });
  </script>
</body>
</html>
```

En este ejemplo, al hacer clic en el botón se descargan los posts y se muestran en la consola del navegador. El método fetch() retorna una **Promise**, que luego procesamos convirtiendo la respuesta a JSON.

Ejemplo 2: Mostrar publicaciones en una lista HTML

HTML (index.html):

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <title>Descarga de datos - Ejemplo 2</title>
</head>
<body>

  <h1>Publicaciones (JSONPlaceholder)</h1>
  <button id="btnMostrar">Mostrar Posts</button>
  <ul id="listaPosts"></ul>

  <script>
    document.getElementById('btnMostrar').addEventListener('click', () => {
      fetch('https://jsonplaceholder.typicode.com/posts')
        .then(respuesta => respuesta.json())
        .then(datos => {
          const lista = document.getElementById('listaPosts');
          lista.innerHTML = ''; // Limpiamos la lista antes de mostrar

          datos.forEach(post => {
            const li = document.createElement('li');
```

```
            li.textContent = `ID: ${post.id} - Título: ${post.title}`;
            lista.appendChild(li);
          });
        })
        .catch(error => {
          console.error("Error al obtener datos:", error);
        });
    });
  </script>

</body>
</html>
```

En este segundo ejemplo se muestra cómo volcar los datos de cada post en un elemento <ul> con elementos <li> para cada uno de los resultados obtenidos.

4. Dos ejemplos de uso básico para subida de datos (POST) con JavaScript y HTML

A continuación, se muestra cómo usar el endpoint de **JSONPlaceholder** para simular el envío de datos (POST). Aunque JSONPlaceholder no guarda realmente los datos enviados, responde como si se hubieran recibido correctamente.

Ejemplo 1: Envío de formulario básico

HTML (index.html):

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <title>Subida de datos - Ejemplo 1</title>
</head>
<body>
  <h1>Enviar Nuevo Post</h1>
  <form id="formPost">
    <label for="titulo">Título:</label>
    <input type="text" id="titulo" name="titulo" required />
    <br><br>
    <label for="cuerpo">Cuerpo:</label>
    <textarea id="cuerpo" name="cuerpo" required></textarea>
    <br><br>
    <button type="submit">Enviar</button>
  </form>

  <script>
    const form = document.getElementById('formPost');
    form.addEventListener('submit', (e) => {
          e.preventDefault();  // Evitamos el comportamiento por defecto del
formulario el cual recargaría la página al realizar el envío

      const postData = {
        title: document.getElementById('titulo').value,
        body: document.getElementById('cuerpo').value,
        userId: 1
      };
```

```
      fetch('https://jsonplaceholder.typicode.com/posts', {
        method: 'POST',
        headers: {
          'Content-type': 'application/json; charset=UTF-8'
        },
        body: JSON.stringify(postData)
      })
      .then(response => response.json())
      .then(data => {
        console.log("Post creado:", data);
        alert(`Se ha simulado la creación del Post con ID: ${data.id}`);
      })
      .catch(error => {
        console.error("Error al crear el Post:", error);
      });
    });
  </script>
</body>
</html>
```

En este ejemplo, se envía la información del formulario como un objeto JSON, indicando las cabeceras adecuadas para el envío. El servidor simulado responderá con un objeto JSON que incluye un id ficticio para el post recién “creado”.

Ejemplo 2: Botón de envío directo

HTML (index.html):

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Subida de datos - Ejemplo 2</title>
</head>
<body>
  <h1>Botón de Envío Directo</h1>
  <button id="btnEnviar">Enviar Post</button>

  <script>
    document.getElementById('btnEnviar').addEventListener('click', () => {
      const nuevoPost = {
        title: "Título de ejemplo",
        body: "Este es el cuerpo del Post de prueba.",
        userId: 99
      };

      fetch('https://jsonplaceholder.typicode.com/posts', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json; charset=UTF-8'
        },
        body: JSON.stringify(nuevoPost)
      })
      .then(res => res.json())
      .then(data => {
        console.log("Post enviado:", data);
        alert("Simulación de envío realizada con éxito");
      })
```

```
      .catch(error => {
        console.error("Error al enviar:", error);
      });
    });
  </script>
</body>
</html>
```

En este ejemplo, los datos se encuentran fijos en el código. Se presiona el botón y se realiza la solicitud **POST** para enviar el nuevo post a JSONPlaceholder.

5. Dos ejemplos de subida y bajada de datos con JavaScript y HTML

A continuación, se presentan dos ejemplos en los que se combinan la descarga (GET) y la subida (POST) de datos.

Ejemplo 1: Lista de posts y formulario de creación en la misma página

HTML (index.html):

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Subida y Bajada de Datos - Ejemplo 1</title>
</head>
<body>
  <h1>Gestión de Posts</h1>

  <!-- Sección para mostrar los Posts -->
  <button id="btnObtenerPosts">Obtener Publicaciones</button>
  <ul id="listaPosts"></ul>

  <!-- Sección para crear un Post -->
  <h2>Crear un Nuevo Post</h2>
  <form id="formCrear">
    <label for="titulo">Título:</label>
    <input type="text" id="titulo" name="titulo" required />
    <br><br>
    <label for="cuerpo">Cuerpo:</label>
    <textarea id="cuerpo" name="cuerpo" required></textarea>
    <br><br>
    <button type="submit">Crear</button>
  </form>

  <script>
    // Obtener Posts
    document.getElementById('btnObtenerPosts').addEventListener('click', () => {
      fetch('https://jsonplaceholder.typicode.com/posts')
        .then(respuesta => respuesta.json())
        .then(datos => {
          const lista = document.getElementById('listaPosts');
          lista.innerHTML = ''; // Limpiar la lista antes de mostrar

          datos.slice(0, 5).forEach(post => {
            // Solo mostramos los primeros 5 resultados
```

```
            const li = document.createElement('li');
            li.textContent = `ID: ${post.id} - Título: ${post.title}`;
            lista.appendChild(li);
          });
        })
        .catch(error => {
          console.error("Error al obtener los posts:", error);
        });
    });

    // Crear un Post
    document.getElementById('formCrear').addEventListener('submit', (e) => {
      e.preventDefault();
      const nuevoPost = {
        title: document.getElementById('titulo').value,
        body: document.getElementById('cuerpo').value,
        userId: 1
      };

      fetch('https://jsonplaceholder.typicode.com/posts', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json; charset=UTF-8'
        },
        body: JSON.stringify(nuevoPost)
      })
      .then(response => response.json())
      .then(data => {
        alert(`Se ha simulado la creación del Post con ID: ${data.id}`);
        console.log("Nuevo Post creado:", data);
      })
      .catch(error => {
        console.error("Error al crear el Post:", error);
      });
    });
  </script>

</body>
</html>
```

En este ejemplo, con un botón se obtienen los posts (solo se muestran 5 para mayor claridad), y en la misma página existe un formulario para crear un nuevo post. Así combinamos GET y POST en un mismo entorno.

Ejemplo 2: Mostrar y crear comentarios en una misma página

HTML (index.html):

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Subida y Bajada de Datos - Ejemplo 2</title>
</head>
<body>
  <h1>Gestión de Comentarios</h1>

  <!-- Sección para mostrar los Comentarios -->
  <button id="btnObtenerComentarios">Obtener Comentarios</button>
  <ul id="listaComentarios"></ul>

  <!-- Sección para crear un Comentario -->
```

```
  <h2>Crear un Nuevo Comentario</h2>
  <form id="formComentario">
    <label for="nombre">Nombre:</label>
    <input type="text" id="nombre" name="nombre" required />
    <br><br>
    <label for="email">Correo:</label>
    <input type="email" id="email" name="email" required />
    <br><br>
    <label for="texto">Comentario:</label>
    <textarea id="texto" name="texto" required></textarea>
    <br><br>
    <button type="submit">Enviar Comentario</button>
  </form>

  <script>
    // Obtener Comentarios
      document.getElementById('btnObtenerComentarios').addEventListener('click',
() => {
      fetch('https://jsonplaceholder.typicode.com/comments')
        .then(response => response.json())
        .then(data => {
          const lista = document.getElementById('listaComentarios');
          lista.innerHTML = '';
          // Mostramos solo los primeros 5 comentarios
          data.slice(0, 5).forEach(comment => {
            const li = document.createElement('li');
               li.textContent = `ID: ${comment.id} - Nombre: ${comment.name} -
Email: ${comment.email}`;
            lista.appendChild(li);
          });
        })
        .catch(error => {
          console.error("Error al obtener los comentarios:", error);
        });
    });

    // Crear Comentario
     document.getElementById('formComentario').addEventListener('submit', (e) =>
{
      e.preventDefault();
      const nuevoComentario = {
        name: document.getElementById('nombre').value,
        email: document.getElementById('email').value,
        body: document.getElementById('texto').value,
        postId: 1
      };

      fetch('https://jsonplaceholder.typicode.com/comments', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json; charset=UTF-8'
        },
        body: JSON.stringify(nuevoComentario)
      })
      .then(res => res.json())
      .then(data => {
        alert(`Se ha simulado el envío del Comentario con ID: ${data.id}`);
        console.log("Comentario enviado:", data);
      })
      .catch(error => {
        console.error("Error al enviar comentario:", error);
      });
    });
  </script>
```

```
</body>
</html>
```

Igual que en el caso anterior, la página permite **descargar** (GET) un conjunto de comentarios y **crear** (POST) un comentario nuevo, mostrando el ID simulado que devuelve la API.

6. Un ejemplo final de una página web con dos zonas: subida de datos y bajada de datos de forma asíncrona

Finalmente, se muestra un ejemplo más completo donde se tiene una **zona de subida** (formulario) y una **zona de bajada** (lista de posts) en la misma vista, con la lógica asíncrona completamente integrada.

HTML (index.html):

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Página Completa - Subida y Bajada de Datos</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 20px;
    }
    .container {
      display: flex;
      gap: 50px;
    }
    .zona {
      border: 1px solid #ccc;
      padding: 20px;
      width: 45%;
      box-sizing: border-box;
    }
    h2 {
      margin-top: 0;
    }
  </style>
</head>
<body>
  <h1>Ejemplo Final: Subida y Bajada Asíncrona de Datos</h1>

  <div class="container">
    <!-- Zona para Subir Datos -->
    <div class="zona">
      <h2>Subir Nuevo Post</h2>
      <form id="formPost">
        <div>
          <label for="titulo">Título:</label><br>
          <input type="text" id="titulo" name="titulo" required />
        </div>
        <br>
        <div>
          <label for="contenido">Contenido:</label><br>
          <textarea id="contenido" name="contenido" required></textarea>
        </div>
        <br>
        <button type="submit">Subir</button>
```

```
      </form>
    </div>

    <!-- Zona para Descargar Datos -->
    <div class="zona">
      <h2>Descargar Posts</h2>
      <button id="btnDescargar">Descargar Lista de Posts</button>
      <ul id="listaPosts"></ul>
    </div>
  </div>

  <script>
    // Referencias a elementos del DOM
    const formPost = document.getElementById('formPost');
    const btnDescargar = document.getElementById('btnDescargar');
    const listaPosts = document.getElementById('listaPosts');

    // Subir datos (POST)
    formPost.addEventListener('submit', (e) => {
      e.preventDefault();
      const nuevoPost = {
        title: document.getElementById('titulo').value,
        body: document.getElementById('contenido').value,
        userId: 1
      };

      fetch('https://jsonplaceholder.typicode.com/posts', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json; charset=UTF-8'
        },
        body: JSON.stringify(nuevoPost)
      })
      .then(response => response.json())
      .then(data => {
        alert(`Simulación de creación del Post con ID: ${data.id}`);
        console.log("Post creado:", data);
      })
      .catch(error => {
        console.error("Error al subir el Post:", error);
      });
    });

    // Descargar datos (GET)
    btnDescargar.addEventListener('click', () => {
      fetch('https://jsonplaceholder.typicode.com/posts')
        .then(response => response.json())
        .then(posts => {
          listaPosts.innerHTML = ''; // Limpiar la lista antes de mostrar
          posts.slice(0, 10).forEach(post => {
            // Mostramos solo los primeros 10
            const li = document.createElement('li');
            li.textContent = `ID: ${post.id} - Título: ${post.title}`;
            listaPosts.appendChild(li);
          });
        })
        .catch(error => {
          console.error("Error al descargar los Posts:", error);
        });
    });
  </script>
</body>
</html>
```

Explicación del funcionamiento:

1. La **zona de subida** contiene un formulario con campos para el título y el contenido. Al enviar el formulario, se construye un objeto JavaScript que se envía vía **fetch()** con el método **POST** a https://jsonplaceholder.typicode.com/posts.
2. La **zona de descarga** incluye un botón que, al hacer clic, realiza una petición **GET** para obtener los posts y los muestra dinámicamente en un listado <ul>.
3. De este modo, en una sola página conviven ambas funcionalidades de manera asíncrona, lo que permite demostrar la interacción con un servicio que maneja JSON sin necesidad de configurar un backend real.

Conclusiones

- **JSONPlaceholder** es una herramienta ideal para probar operaciones asíncronas de subida y bajada de datos con JavaScript.
- Permite simular un entorno de desarrollo similar al de una API REST real, pero sin la complejidad de requerir un servidor o una base de datos.
- Con unos sencillos ejemplos de HTML + JavaScript, se pueden comprender las bases de las peticiones HTTP (GET, POST, etc.) y de la manipulación de datos JSON.

Este manual pretende servir de guía para que puedas iniciarte en el uso de **JSONPlaceholder** y cubrir los primeros pasos del desarrollo web asíncrono con JavaScript. Si se requiere un uso más avanzado, también se pueden explorar otros métodos como **PUT**, **PATCH**, o **DELETE** que JSONPlaceholder ofrece de manera similar.