Entrenamiento – Módulo 3 Semana 3

Hoja de trabajo

El objetivo de esta hoja de trabajo es guiarte en la implementación práctica con **APIs simuladas** y operaciones **CRUD** utilizando JavaScript. Este ejercicio consolidará tus conocimientos sobre **métodos HTTP**, manejo de datos en formato **JSON**, y el uso de **Fetch API** para interactuar con servidores. Al finalizar, habrás creado un programa funcional que se conecta a un servidor local, gestiona datos y maneja errores de manera eficiente.

Problema a resolver: Interacción con Servidores y Consumo de APIs

Tu tarea consiste en desarrollar un programa que:

1. Configure un servidor local utilizando JSON Server, simulando un backend que gestiona una colección de productos.
2. Realice operaciones CRUD (crear, leer, actualizar y eliminar) sobre los datos almacenados en el servidor utilizando Fetch API.
3. Maneje errores y valide las respuestas del servidor para garantizar la robustez de la aplicación.

Pasos sugeridos para solucionar el problema:

Paso 1: Configuración del proyecto

1. Crea un archivo JavaScript llamado **gestión_api.js**.
2. Configura tu entorno de desarrollo:
    - Asegúrate de tener instalado Node.js y JSON Server.
        - npm install -g json-server
        - json-server --version
    - Configura un archivo db.json para actuar como base de datos inicial.

```
{
  "productos": [
    { "id": 1, "nombre": "Laptop", "precio": 1500 },
    { "id": 2, "nombre": "Mouse", "precio": 25 },
    { "id": 3, "nombre": "Teclado", "precio": 50 }
  ]
}

  return go(f, seed, [])
}
```

Sebastián Agudelo. (2024) – Captura de pantalla del código, paso 1.2 entrenamiento – módulo 3 – semana 3

3. Corre **json-server --watch db.json** para que el server ejecute el archivo anteriormente creado.

Paso 2: Operaciones CRUD con Fetch API

1. **Lectura de Datos (GET):** obtén todos los productos almacenados en el servidor y muestra los datos en la consola.
    - Código para solicitud GET:

```
fetch('http://localhost:3000/productos')
  .then(response => response.json())
  .then(data => console.log("Productos disponibles:", data))
  .catch(error => console.error("Error al obtener productos:", error));
```

Sebastián Agudelo. (2024) – Captura de pantalla del código, paso 2.1 entrenamiento – módulo 3 – semana 3

2. **Creación de Nuevos Datos (POST):** añade un nuevo producto a la colección del servidor, validando los datos antes de enviarlos.

```
const nuevoProducto = { id: 4, nombre: "Monitor", precio: 200 };

fetch('http://localhost:3000/productos', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify(nuevoProducto)
})
  .then(response => response.json())
  .then(data => console.log("Producto agregado:", data))
  .catch(error => console.error("Error al agregar producto:", error));
```

Sebastián Agudelo. (2024) – Captura de pantalla del código, paso 2.2 entrenamiento – módulo 3 – semana 3

3. **Actualización de Datos (PUT):** modifica las propiedades de un producto existente.

```
const productoActualizado = { nombre: "Laptop", precio: 1400 };

fetch('http://localhost:3000/productos/1', {
  method: 'PUT',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify(productoActualizado)
})
  .then(response => response.json())
  .then(data => console.log("Producto actualizado:", data))
  .catch(error => console.error("Error al actualizar producto:", error));
```

Sebastián Agudelo. (2024) – Captura de pantalla del código, paso 2.3 entrenamiento – módulo 3 – semana 3

4. **Eliminación de Datos (DELETE):** elimina un producto del servidor basándote en su ID.

```
fetch('http://localhost:3000/productos/3', { method: 'DELETE' })
  .then(() => console.log("Producto eliminado"))
  .catch(error => console.error("Error al eliminar producto:", error));
```

Sebastián Agudelo. (2024) – Captura de pantalla del código, paso 2.4 entrenamiento – módulo 3 – semana 3

Paso 3: Validaciones y Manejo de Errores

1. Implementa validaciones antes de enviar datos al servidor. Por ejemplo:
    - Verifica que todos los campos requeridos estén completos.
    - Asegúrate de que los precios sean valores numéricos válidos.
2. Maneja errores utilizando los métodos de Fetch API (catch) para capturar posibles fallos en las solicitudes.

```
function validarProducto(producto) {
  if (!producto.nombre || typeof producto.precio !== "number") {
    console.error("Datos del producto no válidos.");
    return false;
  }
  return true;
}
```

Sebastián Agudelo. (2024) – Captura de pantalla del código, paso 3 entrenamiento – módulo 3 – semana 3

Paso 4: Pruebas del programa

1. Realiza pruebas con diferentes escenarios:
    - Visualización de datos en la consola después de realizar una solicitud GET.
    - Creación de un producto nuevo y verificación de su adición al servidor.
    - Actualización de un producto existente y confirmación del cambio.
    - Eliminación de un producto y validación de que ya no existe en el servidor.

Paso 5: Ejecución del programa

1. Guarda el archivo **gestión_api.js** en tu computadora.
2. Ejecuta tu programa:
    - Usa Node.js: Ejecuta el archivo con el comando **node gestion_api.js** recuerda que en otra ventana de tu terminal debería estar corriendo **json-server --watch db.json**

Resultado esperado:

Sebastián Agudelo. (2024) – Captura de pantalla del código, paso cierre entrenamiento – módulo 3 – semana 3

1. Gestión eficiente de productos simulados mediante operaciones CRUD.
2. Visualización clara de datos en la consola.
3. Implementación de validaciones y manejo de errores robusto.