Taller: Construyendo una Aplicación CRUD con Supabase

Hoy vamos a continuar con nuestro proyecto en Supabase para crear una aplicación CRUD completa (Crear, Leer, Actualizar, Eliminar). Partiremos desde lo que ya tienen hecho: el listado de productos. Ampliaremos esa funcionalidad para buscar productos por su ID, crear nuevos productos, actualizarlos y eliminarlos.

Al final, habrá un desafío extra para quienes quieran explorar más allá.

Parte 1: Buscar un producto por ID (GET)

1. **¿Qué es un GET por ID?**
   En lugar de obtener todos los productos, podemos pedirle a Supabase que nos devuelva solo el producto con un ID específico. Esto es útil cuando necesitamos información detallada de un solo producto.

2. **Configurar políticas de seguridad (RLS)** Antes de realizar cualquier operación, debemos asegurarnos de que Supabase permita que los usuarios anónimos puedan leer datos:

   - Ve al panel de Supabase.
   - En la tabla **productos**, dirígete a la pestaña **RLS**.
   - Asegúrate de que RLS está habilitado y luego agrega la siguiente política:

     Policy Name: `Allow select for anon users`
     Operation: `SELECT`
     Using Expression: `true`

   Esto permitirá que los usuarios anónimos puedan leer los datos de la tabla.

3. **Cómo se hace?**
   Vamos a usar el método `.select()` como antes, pero esta vez aplicaremos un filtro usando `.eq()` para especificar el ID que queremos.

4. **Código:**
   Añade el siguiente botón y función al proyecto:

   HTML:

   ```
   <input type="number" id="productId" placeholder="ID del producto" />
   <button id="fetchById">Buscar por ID</button>
   ```

   JavaScript (añadir a `script.js`):

   ```
   async function fetchProductById() {
     const productId = document.getElementById("productId").value;
     if (!productId) {
       alert("Por favor, ingresa un ID válido.");
       return;
   ```

```
  }

  try {
    console.log(`Buscando producto con ID: ${productId}...`);
    const { data, error } = await supabase
      .from("productos")
      .select("*")
      .eq("id", productId);

    if (error) {
      console.error("Error al obtener el producto:", error);
      return;
    }

    console.log("Producto obtenido:", data);
    document.getElementById("dataContainer").innerText = JSON.stringify(data,
null, 2);
  } catch (err) {
    console.error("Error en la conexión a Supabase:", err);
  }
}

document.getElementById("fetchById").addEventListener("click",
fetchProductById);
```

Parte 2: Crear un producto (POST)

1. **¿Qué es un POST?**
   En el contexto de HTTP, un método **POST** se usa para enviar datos al servidor y agregar nueva información. En nuestro caso, enviaremos un nuevo producto para que se almacene en la tabla `productos`.

2. **Configurar políticas de seguridad (RLS)**
   Para que los usuarios puedan insertar datos, necesitamos agregar una política:

   - Ve a la pestaña **RLS** de la tabla `productos`.
   - Añade la siguiente política:

     Policy Name: `Allow insert for anon users`
     Operation: `INSERT`
     Using Expression: `true`

   Además, verifica que las columnas necesarias (por ejemplo, `name`, `price`, etc.) estén bien definidas en la tabla.

3. **Cómo se hace?**
   Utilizaremos el método `.insert()` de Supabase.

4. **Código:**
   Añade un formulario para ingresar datos del producto:

   HTML:

```
<h2>Crear un Producto</h2>
<input type="text" id="productName" placeholder="Nombre del producto" />
<input type="number" id="productPrice" placeholder="Precio del producto" />
<button id="createProduct">Crear Producto</button>
```

JavaScript:

```
async function createProduct() {
  const name = document.getElementById("productName").value;
  const price = document.getElementById("productPrice").value;

  if (!name || !price) {
    alert("Por favor, ingresa todos los datos.");
    return;
  }

  try {
    const { data, error } = await supabase
      .from("productos")
      .insert([{ name, price }]);

    if (error) {
      console.error("Error al crear producto:", error);
      return;
    }

    console.log("Producto creado:", data);
    alert("Producto creado exitosamente.");
  } catch (err) {
    console.error("Error al conectar a Supabase:", err);
  }
}

document.getElementById("createProduct").addEventListener("click",
createProduct);
```

Parte 3: Actualizar un producto (PATCH)

1. **¿Qué es un PATCH?**
   El método **PATCH** se usa para modificar datos existentes en el servidor. Por ejemplo, podemos cambiar el nombre o el precio de un producto ya almacenado.

2. **Configurar políticas de seguridad (RLS)**
   Para permitir actualizaciones:

   - Ve a la pestaña **RLS** de la tabla `productos`.
   - Añade la siguiente política:

     Policy Name: `Allow update for anon users`
     Operation: `UPDATE`
     Using Expression: `true`

3. **Cómo se hace?**
   Utilizaremos el método `.update()` de Supabase.

4. **Código:**
   Añade campos y un botón para actualizar un producto:

   HTML:

```
<h2>Actualizar un Producto</h2>
<input type="number" id="updateId" placeholder="ID del producto" />
<input type="text" id="updateName" placeholder="Nuevo nombre" />
<input type="number" id="updatePrice" placeholder="Nuevo precio" />
<button id="updateProduct">Actualizar Producto</button>
```

   JavaScript:

```
async function updateProduct() {
  const id = document.getElementById("updateId").value;
  const name = document.getElementById("updateName").value;
  const price = document.getElementById("updatePrice").value;

  if (!id || (!name && !price)) {
    alert("Por favor, ingresa un ID y al menos un dato para actualizar.");
    return;
  }

  try {
    const { data, error } = await supabase
      .from("productos")
      .update({ name, price })
      .eq("id", id);

    if (error) {
      console.error("Error al actualizar producto:", error);
      return;
    }

    console.log("Producto actualizado:", data);
    alert("Producto actualizado exitosamente.");
  } catch (err) {
    console.error("Error al conectar a Supabase:", err);
  }
}

document.getElementById("updateProduct").addEventListener("click",
updateProduct);
```

Parte 4: Eliminar un producto (DELETE)

1. **¿Qué es un DELETE?**
   Este método elimina un recurso específico del servidor. En este caso,

eliminaremos un producto según su ID.

2. **Configurar políticas de seguridad (RLS)**
   Para permitir eliminaciones:
   - Ve a la pestaña **RLS** de la tabla `productos`.
   - Añade la siguiente política:

     Policy Name: `Allow delete for anon users`
     Operation: `DELETE`
     Using Expression: `true`

3. **Cómo se hace?**
   Utilizaremos el método `.delete()` de Supabase.

4. **Código:**
   Añade un campo y un botón para eliminar un producto:

   HTML:

```html
<h2>Eliminar un Producto</h2>
<input type="number" id="deleteId" placeholder="ID del producto" />
<button id="deleteProduct">Eliminar Producto</button>
```

   JavaScript:

```javascript
async function deleteProduct() {
  const id = document.getElementById("deleteId").value;

  if (!id) {
    alert("Por favor, ingresa un ID válido.");
    return;
  }

  try {
    const { data, error } = await supabase
      .from("productos")
      .delete()
      .eq("id", id);

    if (error) {
      console.error("Error al eliminar producto:", error);
      return;
    }

    console.log("Producto eliminado:", data);
    alert("Producto eliminado exitosamente.");
  } catch (err) {
    console.error("Error al conectar a Supabase:", err);
  }
}
```

```
document.getElementById("deleteProduct").addEventListener("click",
deleteProduct);
```

Desafío Extra: Búsqueda avanzada y ordenamiento

Para los estudiantes que quieran explorar más allá:

1. **Búsqueda por rango de precios:**
   Permite buscar productos cuyo precio esté entre un rango definido por el usuario.

   HTML:

   ```
   <h2>Buscar productos por rango de precios</h2>
   <input type="number" id="minPrice" placeholder="Precio mínimo" />
   <input type="number" id="maxPrice" placeholder="Precio máximo" />
   <button id="searchByPrice">Buscar</button>
   ```

   JavaScript:

   ```
   async function searchByPriceRange() {
     const minPrice = document.getElementById("minPrice").value;
     const maxPrice = document.getElementById("maxPrice").value;

     if (!minPrice || !maxPrice) {
       alert("Por favor, ingresa ambos precios.");
       return;
     }

     try {
       const { data, error } = await supabase
         .from("productos")
         .select("*")
         .gte("price", minPrice)
         .lte("price", maxPrice);

       if (error) {
         console.error("Error al buscar productos:", error);
         return;
       }

       console.log("Productos encontrados:", data);
       document.getElementById("dataContainer").innerText = JSON.stringify(data,
   null, 2);
     } catch (err) {
       console.error("Error en la conexión a Supabase:", err);
     }
   }

   document.getElementById("searchByPrice").addEventListener("click",
   searchByPriceRange);
   ```

2. **Ordenar productos por precio:**
   Experimenta con la función `.order()` de Supabase para mostrar los productos en orden ascendente o descendente por precio.