



Actividad 3 - Funcionamiento del carrito

Desarrollo de Sistemas Web II

Ingeniería en Desarrollo de Software

Tutor: Aarón Iván Salazar Macías

Alumno: Jusi Ismael Linares Gutiérrez

Fecha: 19/02/2024

Índice

Introducción	3
Descripción	3
Justificación	3
Desarrollo:	4
Codificación.....	4
Pruebas del sitio web	14
Conclusión.....	19
GitHub	19

Introducción

En el contexto del desarrollo del proyecto para la tienda online de Sara, se ha avanzado en la implementación del sitio web y se ha aprobado el diseño del mismo. Ahora, el siguiente paso es dotar de funcionalidad al carrito de compras, lo que implica la creación de un API REST para gestionar las operaciones de alta, eliminación, consulta y modificación de los productos en la base de datos.

Este API REST será fundamental para interactuar con la base de datos y permitirá a los usuarios del sitio web realizar diversas acciones relacionadas con los productos disponibles en la tienda. Las operaciones de alta permitirán agregar nuevos productos al inventario, mientras que las operaciones de eliminación permitirán eliminar productos que ya no estén disponibles o que hayan sido descontinuados.

Por otro lado, las operaciones de consulta serán útiles para que los usuarios puedan buscar y visualizar información detallada sobre los productos, como su descripción, precio, disponibilidad, entre otros. Finalmente, las operaciones de modificación permitirán actualizar la información de los productos, como su precio, cantidad disponible, categoría, entre otros aspectos.

Descripción

El contexto presentado describe la solicitud de la tienda Sara para desarrollar la funcionalidad del carrito de compras en su sitio web. La tienda ha aprobado el diseño del sitio web y ahora busca implementar un sistema que permita a los usuarios agregar, eliminar, consultar y modificar productos en su carrito de compras. Esto requerirá el desarrollo de una API REST que interactúe con la base de datos para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en los productos.

La funcionalidad del carrito de compras permitirá a los usuarios seleccionar productos y agregarlos al carrito, donde podrán ver un resumen de los productos seleccionados y la cantidad total de la compra.

El objetivo principal de esta actividad es completar el desarrollo de la tienda en línea de Sara, brindando a los usuarios una experiencia de compra completa y funcional. Al implementar esta funcionalidad, la tienda Sara podrá ofrecer a sus clientes una plataforma en línea donde puedan explorar, seleccionar y comprar productos de manera conveniente y eficiente.

Justificación

La utilización de una API REST para gestionar las operaciones de alta, eliminación, consulta y modificación de productos en la tienda en línea de Sara es una elección estratégica y eficiente por varias razones.

En primer lugar, una API REST proporciona una interfaz estandarizada y basada en estándares web, lo que facilita la integración con diferentes sistemas y plataformas. Esto significa que la tienda en línea de Sara puede interactuar con otros sistemas de manera más sencilla y eficiente, como sistemas de gestión de inventario, sistemas de pago, o sistemas de análisis de datos.

la utilización de una API REST para gestionar los productos de la tienda en línea de Sara ofrece una serie de beneficios clave, incluyendo una integración más fácil con otros sistemas, una arquitectura flexible y escalable, y una mejor separación de preocupaciones. Estas ventajas ayudarán a Sara a desarrollar una tienda en línea robusta, eficiente y adaptable a las necesidades cambiantes del mercado.

Desarrollo:

Codificación

Tomando nuevamente nuestra plantilla empezaremos con el formulario donde

`<form action="http://localhost:9000/api/v1/producto/save" method="POST" enctype="multipart/form-data">`: Inicia un formulario que envía los datos a la URL especificada (`http://localhost:9000/api/v1/producto/save`) usando el método POST y codifica los datos como `multipart/form-data`. Tras esto solo son labels, inputs y un botón para registrar los datos ingresados por el usuario

```
<!-- Formulario para añadir un registro -->
<div class="container mt-5">
  <div class="row justify-content-center">
    <div class="col-lg-6">
      <h2 class="text-center mb-4">Agregar Producto</h2>
      <form action="http://localhost:9000/api/v1/producto/save" method="POST" enctype="multipart/form-data">
        <div class="form-group">
          <label for="Descripcion">Descripción:</label>
          <input type="text" class="form-control" id="Descripcion" name="Descripcion" required>
        </div>
        <div class="form-group">
          <label for="Precio">Precio:</label>
          <input type="number" class="form-control" id="Precio" name="Precio" required>
        </div>
        <div class="form-group">
          <label for="Cantidad">Cantidad:</label>
          <input type="number" class="form-control" id="Cantidad" name="Cantidad" required>
        </div>
        <div class="form-group">
          <label for="idCategorias">ID Categorias:</label>
          <input type="number" class="form-control" id="idCategorias" name="idCategorias" required>
        </div>
        <div class="form-group">
          <label for="idMarcas">ID Marcas:</label>
          <input type="number" class="form-control" id="idMarcas" name="idMarcas" required>
        </div>
        <div class="form-group">
          <label for="img">Imagen:</label>
          <input type="text" class="form-control" id="img" name="img">
        </div>
        <button type="submit" class="btn btn-primary btn-block">Agregar Producto</button>
      </form>
    </div>
  </div>
</div>
```

1. Definimos una función llamada submitForm() que se ejecutará cuando se envíe el formulario.
2. Dentro de la función, creamos un objeto formData que contiene los valores de los campos del formulario.
3. Utilizamos la función fetch() para enviar una solicitud POST a la URL especificada (<http://localhost:9000/api/v1/producto/save>) con los datos del formulario.
4. Configuramos la solicitud POST con el método 'POST' y el tipo de contenido 'application/json'.
5. Convertimos el objeto formData a formato JSON utilizando JSON.stringify() y lo adjunta como cuerpo de la solicitud.
6. Una vez que completamos la solicitud, convierte la respuesta a formato JSON utilizando .json().
7. Se muestra un mensaje de alerta con el mensaje devuelto por la API REST.
8. Manejamos cualquier error que pueda ocurrir durante la solicitud utilizando .catch() y mostramos un mensaje de error en la consola.

```
111 <script>
112     function submitForm() {
113         var formData = {
114             "ID": document.getElementById("ID").value,
115             "Descripcion": document.getElementById("descripcion").value,
116             "Precio": document.getElementById("precio").value,
117             "Cantidad": document.getElementById("cantidad").value,
118             "idCategorias": document.getElementById("idCategorias").value,
119             "idMarcas": document.getElementById("idMarcas").value,
120             "img": document.getElementById("img").value
121         };
122
123         fetch('http://localhost:9000/api/v1/producto/save', {
124             method: 'POST',
125             headers: {
126                 'Content-Type': 'application/json'
127             },
128             body: JSON.stringify(formData)
129         })
130         .then(response => response.json())
131         .then(data => {
132             alert(data.message);
133         })
134         .catch(error => {
135             console.log(error);
136         });
137     }
138 }
```

```

123     fetch('http://localhost:9000/api/v1/producto/save', {
124         method: 'POST',
125         headers: {
126             'Content-Type': 'application/json'
127         },
128         body: JSON.stringify(formData)
129     })
130     .then(response => response.json())
131     .then(data => {
132         alert(data.message);
133     })
134     .catch(error => console.error('Error:', error));
135 }
136 </script>
137 <!--Fin formulario-->

```

Ahora para el Read.html igualmente creamos un formulario con el atributo id establecido como "searchForm". onsubmit="return searchProduct()": Este onsubmit especifica una acción que se llevará a cabo cuando el formulario se envíe, lo veremos mas adelante.

```

<!-- Formulario para Leer un registro -->
<div class="container mt-5">
    <div class="row justify-content-center">
        <div class="col-lg-6">
            <h2 class="text-center mb-4">Buscar Producto</h2>
            <form id="searchForm" onsubmit="return searchProduct()">
                <div class="form-group">
                    <label for="id">ID:</label>
                    <input type="number" class="form-control" id="id" name="id" required>
                </div>
                <button type="submit" class="btn btn-primary btn-block">Buscar</button>
            </form>
        </div>
    </div>
</div>
<!-- Fin formulario -->

```

```
107 <!-- producto devuelto -->
108 <div class="container-fluid pt-5 pb-3">
109   <h2 class="section-title position-relative text-uppercase mx-xl-5 mb-4"><span class="bg-secondary pr-3">Visualización de productos por ID</span></h2>
110   <div class="row px-xl-5" id="productContainer"></div>
111 </div>
112 <!-- Fin producto devuelto -->
```

Se utiliza la función `fetch()` para realizar una solicitud GET a la API REST que busca el producto por su ID.

En el primer `then()`, se convierte la respuesta en formato JSON para obtener los datos del producto.

Si no se encuentra ningún producto con el ID especificado, se muestra una alerta indicando que el producto no fue encontrado.

```

114 <!-- Script para visualizar el producto -->
115 <script>
116     function searchProduct() {
117         var productId = document.getElementById("id").value;
118         fetch('/api/v1/producto/productoById/' + productId)
119             .then(response => response.json())
120             .then(product => {
121                 if (product.length > 0) {
122                     var productHtml = `
123                         <div class="col-lg-3 col-md-4 col-sm-6 pb-1">
124                             <div class="product-item bg-light mb-4">
125                                 <div class="product-img position-relative overflow-hidden">
126                                     
127                                 </div>
128                                 <div class="text-center py-4">
129                                     <a class="h6 text-decoration-none text-truncate" href="${product[0].descripcion}.html">${product[0].descripcion}</a>
130                                     <div class="d-flex align-items-center justify-content-center mt-2">
131                                         <h5>$$${product[0].precio}</h5>
132                                     </div>
133                                     <div class="d-flex align-items-center justify-content-center mb-1">
134                                         <small class="fa fa-star text-primary mr-1"></small>
135                                         <small class="fa fa-star text-primary mr-1"></small>
136                                         <small class="fa fa-star text-primary mr-1"></small>
137                                         <small class="fa fa-star-half-alt text-primary mr-1"></small>
138                                         <small class="far fa-star text-primary mr-1"></small>
139                                         <small>(99)</small>
140                                     </div>
141                                 </div>
142                             </div>
143                         </div>`;
144                     document.getElementById("productContainer").innerHTML = productHtml;
145                 } else {
146                     alert("Producto no encontrado");
147                 }
148             })
149             .catch(error => console.error('Error:', error));
150     return false; // Evita que el formulario se envíe
151 }
152 </script>

```

<form onsubmit="submitForm(); return false;">: Este es el elemento <form> que envuelve los elementos del formulario. El atributo onsubmit indica que cuando se envíe el formulario, se llamará a la función submitForm() para manejar la acción de envío. El return false; asegura que el formulario no se envíe de manera predeterminada cuando se hace clic en el botón de enviar.

Por lo demás son solamente labels, inputs y botones, pero esta vez en los inputs obtendremos los resultados de la api productoById

```
92 <!--Formulario para actualizar un registro-->
93 <div class="container mt-5">
94   <div class="row justify-content-center">
95     <div class="col-lg-6">
96       <h2 class="text-center mb-4">Actualizar Producto</h2>
97       <form onsubmit="submitForm(); return false;">
98         <div class="form-group">
99           <label>ID del Producto a Buscar:</label>
100           <input type="number" class="form-control" id="searchID" name="searchID" required>
101         </div>
102         <button type="button" onclick="getProductById()" class="btn btn-primary btn-block">Buscar
103           Producto</button>
104         <div id="productDetails" style="display: none;">
105           <div class="form-group">
106             <label>ID del Producto:</label>
107             <input type="number" class="form-control" id="ID" name="ID" readonly>
108           </div>
109           <div class="form-group">
110             <label for="Descripcion">Descripción:</label>
111             <input type="text" class="form-control" id="Descripcion" name="Descripcion" required>
112           </div>
113           <div class="form-group">
114             <label for="Precio">Precio:</label>
115             <input type="number" class="form-control" id="Precio" name="Precio" required>
116           </div>
117           <div class="form-group">
118             <label for="Cantidad">Cantidad:</label>
119             <input type="number" class="form-control" id="Cantidad" name="Cantidad" required>
120           </div>
121           <div class="form-group">
122             <label for="idCategorias">ID Categorias:</label>
123             <input type="number" class="form-control" id="idCategorias" name="idCategorias" required>
124           </div>
125           <div class="form-group">
126             <label for="idMarcas">ID Marcas:</label>
127             <input type="number" class="form-control" id="idMarcas" name="idMarcas" required>
128           </div>
129           <div class="form-group">
130             <label for="img">Imagen:</label>
131             <input type="text" class="form-control" id="img" name="img">
132           </div>
133           <button type="submit" class="btn btn-primary btn-block">Actualizar Producto</button>
134         </div>
135       </form>
136     </div>
137   </div>
138 </div>
```


getProductById(): Esta función se llama cuando se hace clic en el botón de búsqueda de producto. Su objetivo es obtener los detalles del producto utilizando el ID proporcionado por el usuario.

var productId = document.getElementById("searchID").value;: Esto obtiene el valor ingresado por el usuario en el campo de entrada con el ID "searchID"

fetch(http://localhost:9000/api/v1/producto/productoById2/\${productId})`: Esta línea realiza una solicitud HTTP GET a la URL proporcionada, que incluye el ID del producto deseado

if (data.length > 0) { ... }: Verifica si se encontró algún producto. Si la longitud de los datos es mayor que cero, significa que se encontró al menos un producto con el ID proporcionado.

var producto = data[0];: Se extrae el primer producto de la respuesta, ya que se espera que la respuesta contenga solo un producto con el ID único.

document.getElementById("ID").value = producto.id;: Asigna el ID del producto encontrado al campo de entrada con el ID "ID" en el formulario.

El código de arriba se repite para los campos descripción, precio, idcategorias, idmarcas e img

```
140 <script>
141     function getProductById() {
142         var productId = document.getElementById("searchID").value;
143         fetch(`http://localhost:9000/api/v1/producto/productoById2/${productId}`)
144             .then(response => response.json())
145             .then(data => {
146                 if (data.length > 0) {
147                     var producto = data[0];
148                     document.getElementById("ID").value = producto.id;
149                     document.getElementById("Descripcion").value = producto.descripcion;
150                     document.getElementById("Precio").value = producto.precio;
151                     document.getElementById("Cantidad").value = producto.cantidad;
152                     document.getElementById("idCategorias").value = producto.idCategorias;
153                     document.getElementById("idMarcas").value = producto.idMarcas;
154                     document.getElementById("img").value = producto.img;
155                     document.getElementById("productDetails").style.display = "block";
156                 } else {
157                     alert("Producto no encontrado");
158                 }
159             })
160             .catch(error => console.error('Error:', error));
161     }
```

Se obtienen los valores de varios campos de entrada del formulario, como el ID, la descripción, el precio, la cantidad, etc.

Se crea un objeto llamado formData que contiene todos los datos obtenidos del formulario.

Se realiza una solicitud HTTP PUT al servidor utilizando la función fetch(). La URL de la solicitud es `http://localhost:9000/api/v1/producto/update`, que es la ruta de la API REST utilizada para actualizar un producto.

Se incluyen los datos del formulario en el cuerpo de la solicitud como un objeto JSON utilizando el método `JSON.stringify()`. Esto convierte el objeto formData en una cadena JSON que puede ser enviada al servidor.

Se especifica que el tipo de contenido de la solicitud es JSON en el encabezado de la solicitud.

Una vez que se envían los datos al servidor, se espera una respuesta. La respuesta del servidor es manejada en dos pasos utilizando las funciones `.then()`.

En el primer `.then()`, se convierte la respuesta del servidor en formato JSON utilizando el método `.json()`.

En el segundo `.then()`, se procesa la respuesta JSON del servidor. En este caso, se muestra un mensaje de alerta al usuario con el mensaje proporcionado por el servidor.

Si hay algún error durante la solicitud o el procesamiento de la respuesta, se captura y se muestra en la consola del navegador utilizando el método `.catch()`

```
162     function submitForm() {
163         var formData = {
164             "ID": document.getElementById("ID").value,
165             "Descripcion": document.getElementById("Descripcion").value,
166             "Precio": document.getElementById("Precio").value,
167             "Cantidad": document.getElementById("Cantidad").value,
168             "idCategorias": document.getElementById("idCategorias").value,
169             "idMarcas": document.getElementById("idMarcas").value,
170             "img": document.getElementById("img").value
171         };
172         fetch(`http://localhost:9000/api/v1/producto/update`, {
173             method: 'PUT',
174             headers: {
175                 'Content-Type': 'application/json'
176             },
177             body: JSON.stringify(formData)
178         })
179             .then(response => response.json())
180             .then(data => {
181                 alert(data.message);
182             })
183             .catch(error => console.error('Error:', error));
184     }
185     </script>
```

Por ultimo pasemos a nuestro Delete.html, aquí utilizamos de base nuestro Update.html para ver que articulo estamos por borrar

```
92 <!--Formulario para eliminar un registro-->
93 <div class="container mt-5">
94   <div class="row justify-content-center">
95     <div class="col-lg-6">
96       <h2 class="text-center mb-4">Eliminar Producto</h2>
97       <form onsubmit="submitForm(); return false;">
98         <div class="form-group">
99           <label>ID del Producto a Buscar:</label>
100           <input type="number" class="form-control" id="searchID" name="searchID" required>
101         </div>
102         <button type="button" onclick="getProductById()" class="btn btn-primary btn-block">Buscar Producto</button>
103         <div id="productDetails" style="display: none;">
104           <div class="form-group">
105             <label>ID del Producto:</label>
106             <input type="number" class="form-control" id="ID" name="ID" readonly>
107           </div>
108           <div class="form-group">
109             <label for="Descripcion">Descripción:</label>
110             <input type="text" class="form-control" id="Descripcion" name="Descripcion" required>
111           </div>
112           <div class="form-group">
113             <label for="Precio">Precio:</label>
114             <input type="number" class="form-control" id="Precio" name="Precio" required>
115           </div>
116           <div class="form-group">
117             <label for="Cantidad">Cantidad:</label>
118             <input type="number" class="form-control" id="Cantidad" name="Cantidad" required>
119           </div>
120           <div class="form-group">
121             <label for="idCategorias">ID Categorias:</label>
122             <input type="number" class="form-control" id="idCategorias" name="idCategorias" required>
123           </div>
124           <div class="form-group">
125             <label for="idMarcas">ID Marcas:</label>
126             <input type="number" class="form-control" id="idMarcas" name="idMarcas" required>
127           </div>
128           <div class="form-group">
129             <label for="img">Imagen:</label>
130             <input type="text" class="form-control" id="img" name="img">
131           </div>
132         <!-- Cambio el botón de actualizar por eliminar -->
133         <button type="button" onclick="deleteProduct()" class="btn btn-danger btn-block">Eliminar Producto</button>
134       </form>
135     </div>
136   </div>
137 </div>
138 </div>
```

El getProductByID funciona igual que el anterior

```
140 <script>
141   function getProductById() {
142     var productId = document.getElementById("searchID").value;
143     fetch(`http://localhost:9000/api/v1/producto/productoById2/${productId}`)
144       .then(response => response.json())
145       .then(data => {
146         if (data.length > 0) {
147           var producto = data[0];
148           document.getElementById("ID").value = producto.id;
149           document.getElementById("Descripcion").value = producto.descripcion;
150           document.getElementById("Precio").value = producto.precio;
151           document.getElementById("Cantidad").value = producto.cantidad;
152           document.getElementById("idCategorias").value = producto.idCategorias;
153           document.getElementById("idMarcas").value = producto.idMarcas;
154           document.getElementById("img").value = producto.img;
155           document.getElementById("productDetails").style.display = "block";
156         } else {
157           alert("Producto no encontrado");
158         }
159       })
160       .catch(error => console.error('Error:', error));

```

Se obtiene el valor del ID del producto que se va a eliminar del campo de entrada con el ID "ID" en el formulario.

Se realiza una solicitud HTTP GET al servidor utilizando la función `fetch()`. La URL de la solicitud es `http://localhost:9000/api/v1/producto/delete/${id}`, donde `${id}` es el ID del producto que se va a eliminar.

Se especifica que el método de la solicitud es GET.

Una vez que se envía la solicitud al servidor, se espera una respuesta. La respuesta del servidor es manejada en dos pasos utilizando las funciones `.then()`.

En el primer `.then()`, se convierte la respuesta del servidor en formato JSON utilizando el método `.json()`.

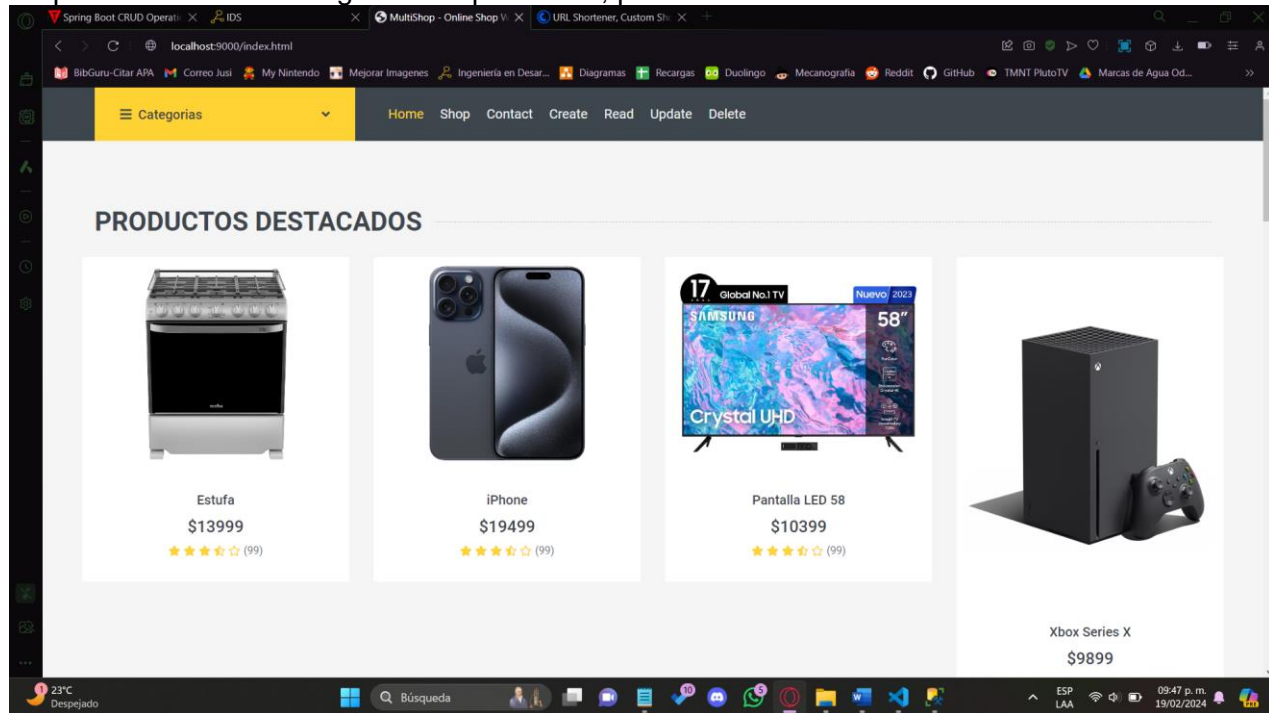
En el segundo `.then()`, se procesa la respuesta JSON del servidor. En este caso, se muestra un mensaje de alerta al usuario con el mensaje proporcionado por el servidor.

Además, se limpian los campos del formulario y se oculta el contenedor de detalles del producto después de eliminar el producto con éxito.

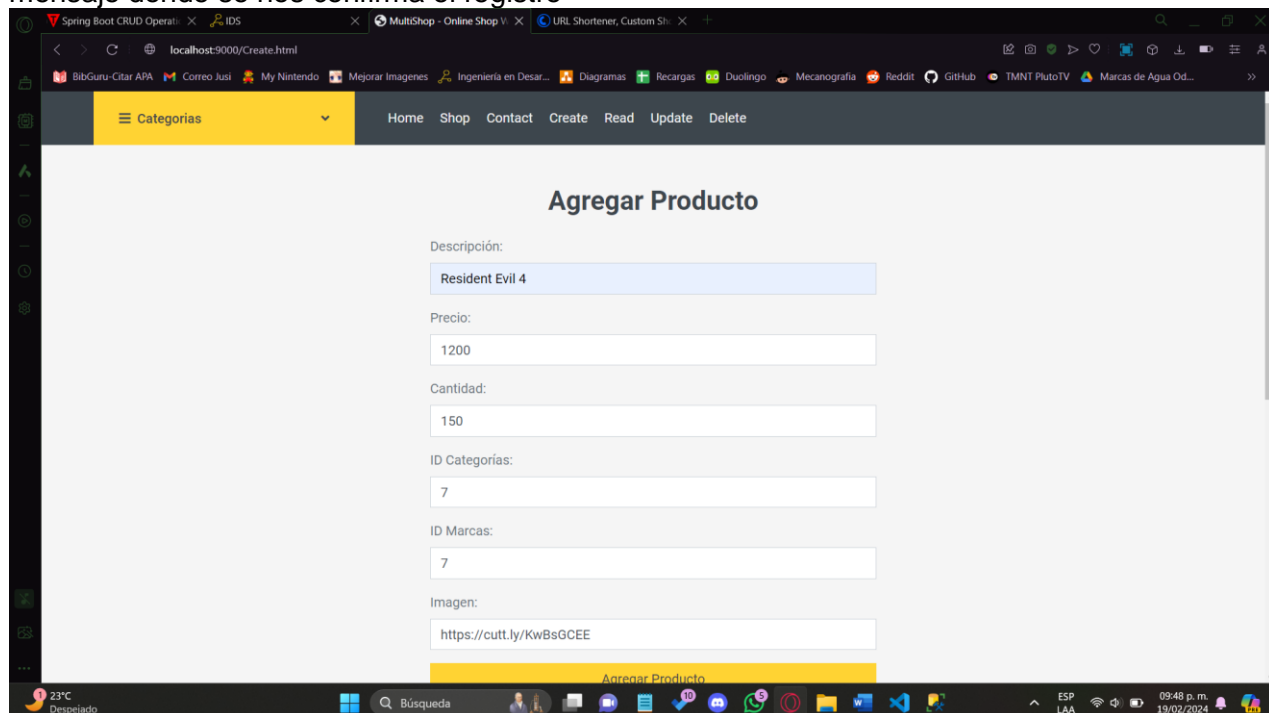
```
163     function deleteProduct() {
164         var id = document.getElementById("ID").value;
165         fetch(`http://localhost:9000/api/v1/producto/delete/${id}`, {
166             method: 'GET'
167         })
168         .then(response => response.json())
169         .then(data => {
170             alert(data.message);
171             // Limpiar los campos después de eliminar el producto
172             document.getElementById("ID").value = "";
173             document.getElementById("Descripcion").value = "";
174             document.getElementById("Precio").value = "";
175             document.getElementById("Cantidad").value = "";
176             document.getElementById("idCategorias").value = "";
177             document.getElementById("idMarcas").value = "";
178             document.getElementById("img").value = "";
179             document.getElementById("productDetails").style.display = "none";
180         })
181         .catch(error => console.error('Error:', error));
182     }
```

Pruebas del sitio web

Empecemos con crear registros de productos, para esto nos vamos a Create.html desde el index



Ingresamos los datos que se nos solicitan y damos clic en agregar producto, nos saldrá un mensaje donde se nos confirma el registro



```
{"succes":null,"message":"Item Saved with success"}
```

Podemos verificar dirigiéndonos al Read.html e ingresando el ID 12 (el 11 fue utilizado para pruebas incluyendo el ser eliminado)

Spring Boot CRUD Operati...IDSMultiShop - Online Shop V...URL Shortener, Custom Sh...

localhost:9000/Read.html

BibGuru-Citar APACorreo JustoMy NintendoMejorar ImagenesIngeniería en Desar...DiagramasRecargasDuolingoMecanografíaRedditGitHubTMNT PlutoTVMarcas de Agua Od...

CategoríasHomeShopContactCreateReadUpdateDelete

Home / Shop / Shop Detail

Buscar Producto

ID:Buscar

VISUALIZACIÓN DE PRODUCTOS POR ID

PONGASE EN CONTACTOBOLETIN

23°CDespejado

Búsqueda

Spring Boot CRUD Operati...IDSMultiShop - Online Shop V...URL Shortener, Custom Sh...

localhost:9000/Read.html


BibGuru-Citar APACorreo JustoMy NintendoMejorar ImagenesIngeniería en Desar...DiagramasRecargasDuolingoMecanografíaRedditGitHubTMNT PlutoTVMarcas de Agua Od...

123

Buscar

VISUALIZACIÓN DE PRODUCTOS POR ID

Xbox Series X

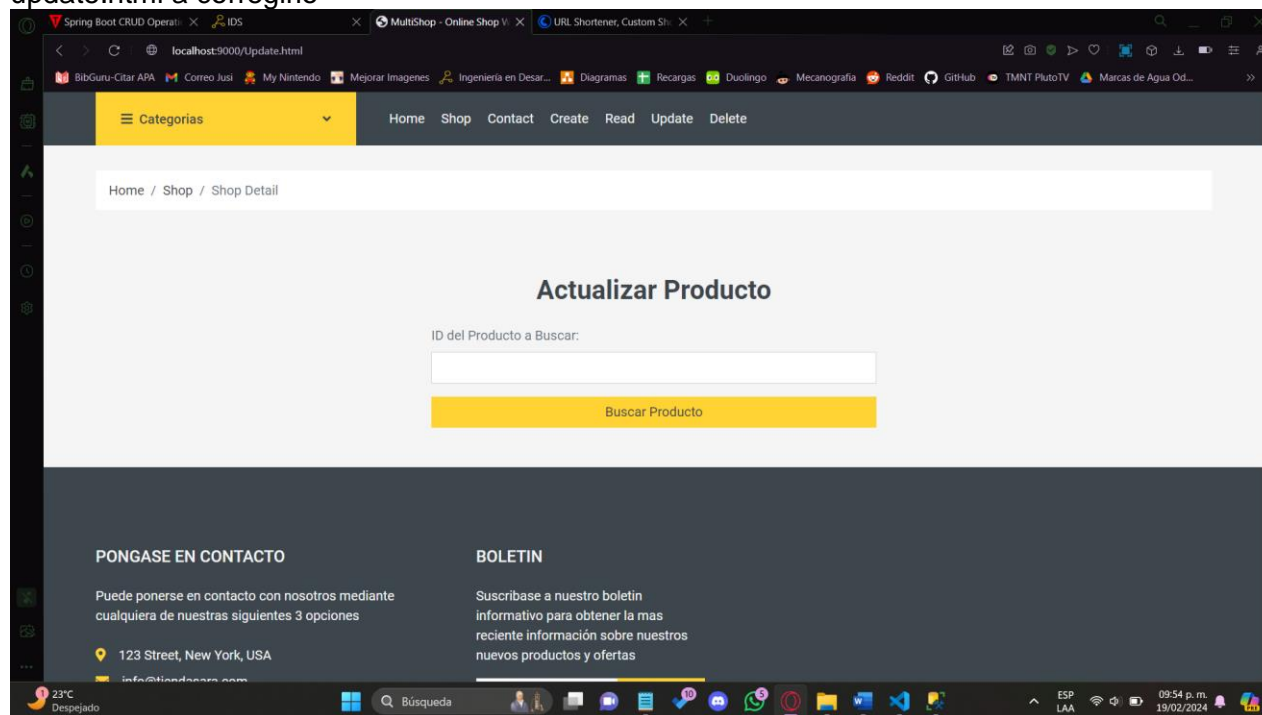


Resident Evil 4

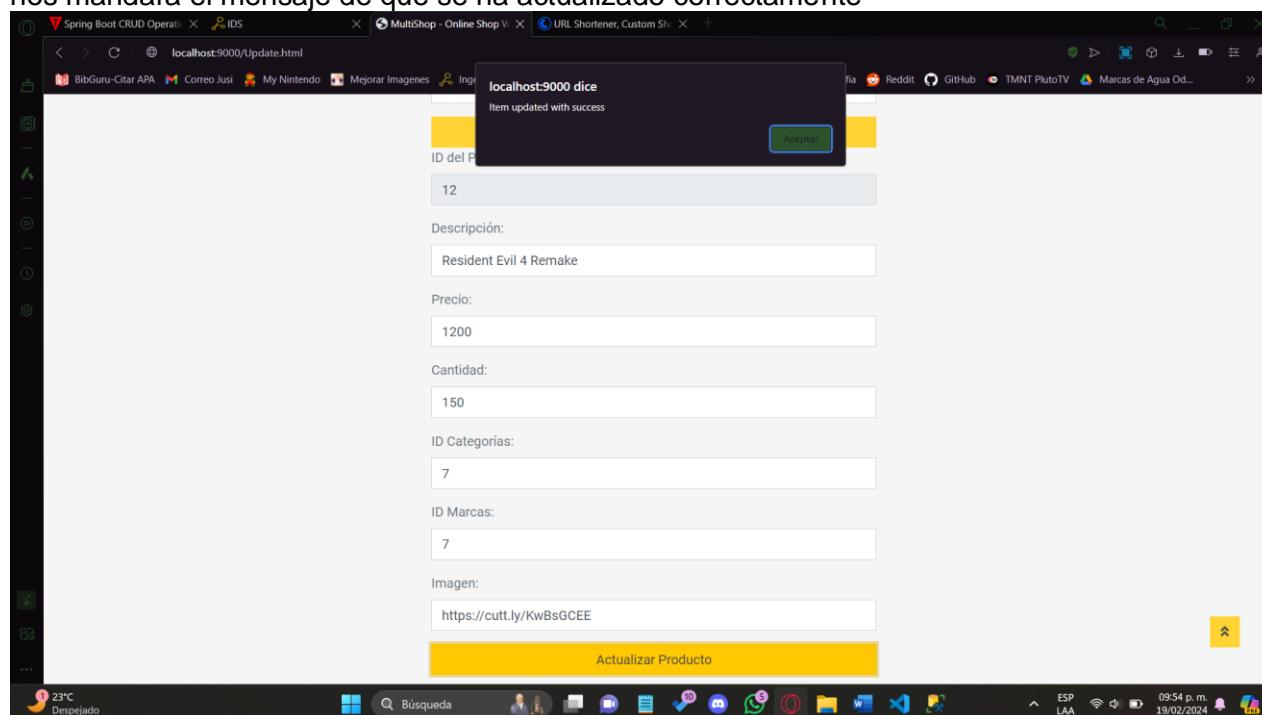
\$1200

★★★★☆ (99)

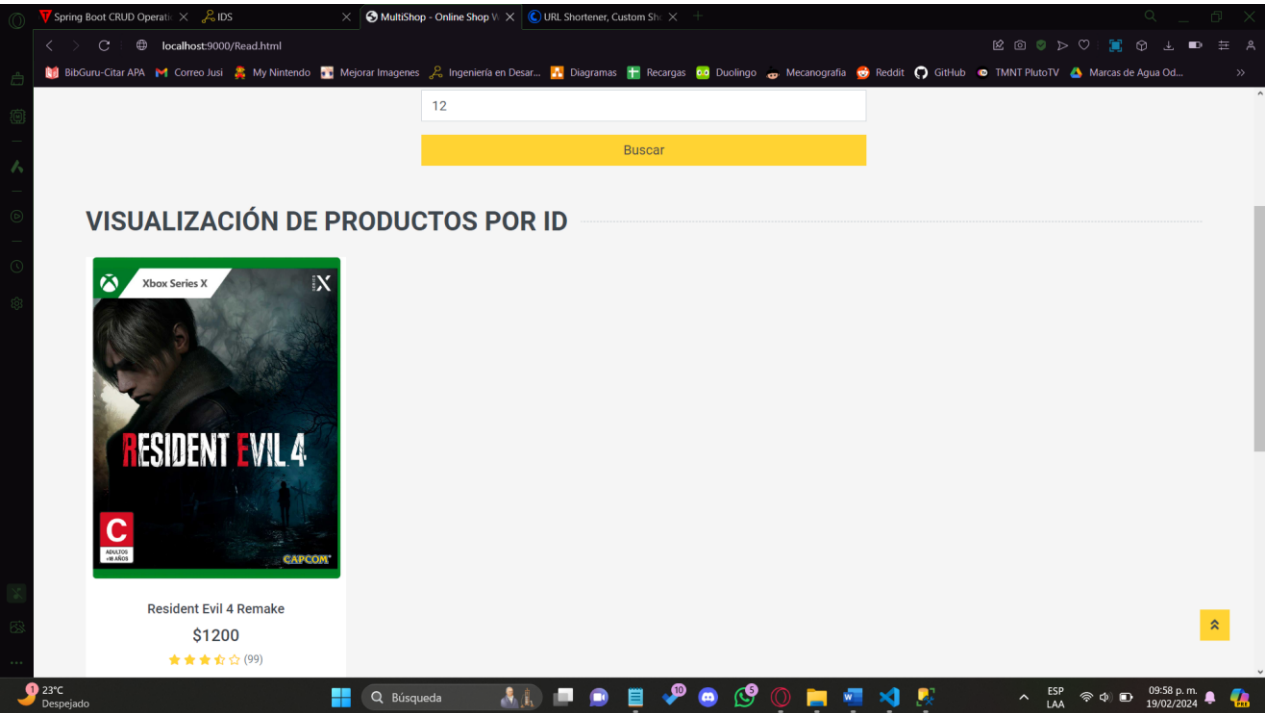
Para este producto cometí un error a propósito, este videojuego es mejor conocido como Resident Evil 4 Remake para evitar confusiones con su versión original, nos dirigiremos para el update.html a corregirlo



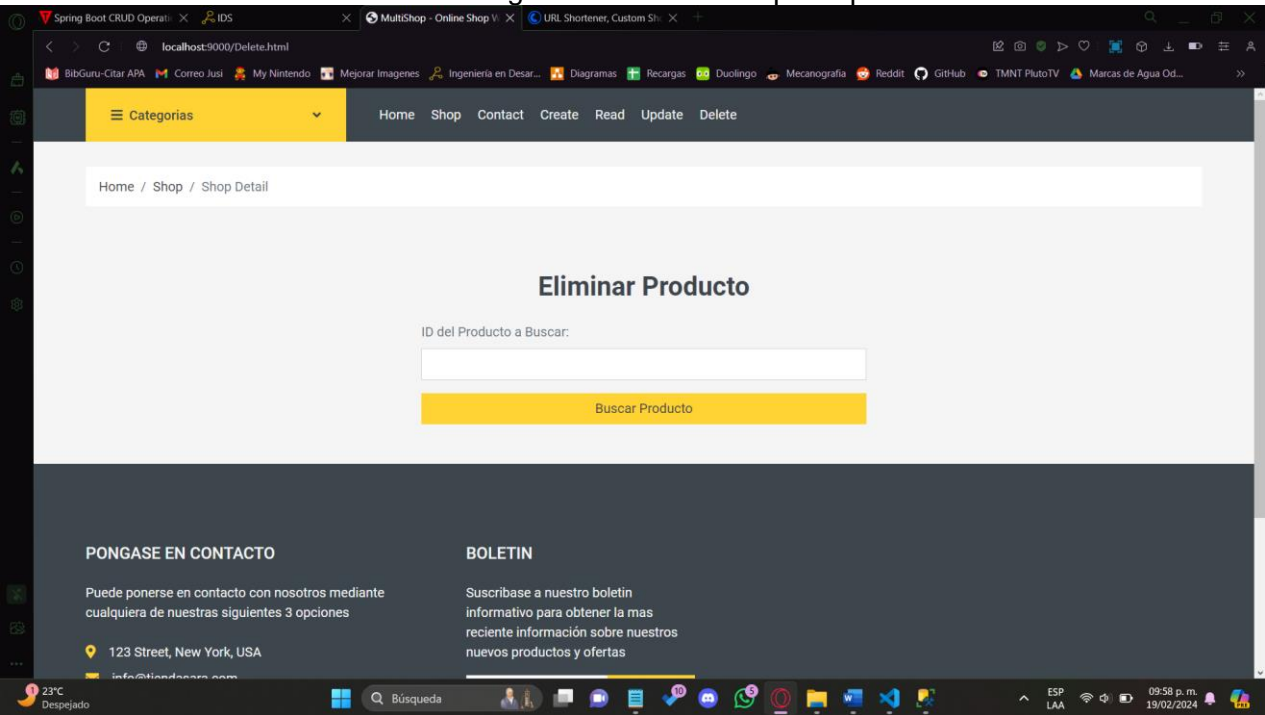
Al ingresar el id nos devolverá todos los datos correspondientes al producto, donde podemos empezar a modificar todo menos el ID, una vez hecha la corrección daremos clic en actualizar y nos mandara el mensaje de que se ha actualizado correctamente



Podemos verificarlo nuevamente en el read.html



Ahora en el Delete.html nuevamente ingresaremos el id 12 para que se elimine el artículo



Spring Boot CRUD Operati... x IDS x MultiShop - Online Shop Vi... x URL Shortener, Custom Sh... x

localhost:9000/Delete.html

BibGuru-Citar APA Correo Justo My Nintendo Mejorar Imagenes Ingeniería en Desar... Diagramas Recargas Duolingo Mecanografía Reddit GitHub TMNT PlutoTV Marcas de Agua Od...

Buscar Producto

ID del Producto:

12

Descripción:

Resident Evil 4 Remake

Precio:

1200

Cantidad:

150

ID Categorías:

7

ID Marcas:

7

Imagen:

https://cutt.ly/KwBsGCEE

Eliminar Producto

23°C Despejado

Búsqueda

Spring Boot CRUD Operati... x IDS x MultiShop - Online Shop Vi... x URL Shortener, Custom Sh... x

localhost:9000/Delete.html

BibGuru-Citar APA Correo Justo My Nintendo Ingeniería en Desar... Diagramas Recargas Duolingo Mecanografía Reddit GitHub TMNT PlutoTV Marcas de Agua Od...

localhost:9000 dice
Item removed with success

Aceptar

ID del P

12

Descripción:

Resident Evil 4 Remake

Precio:

1200

Cantidad:

150

ID Categorías:

7

ID Marcas:

7

Imagen:

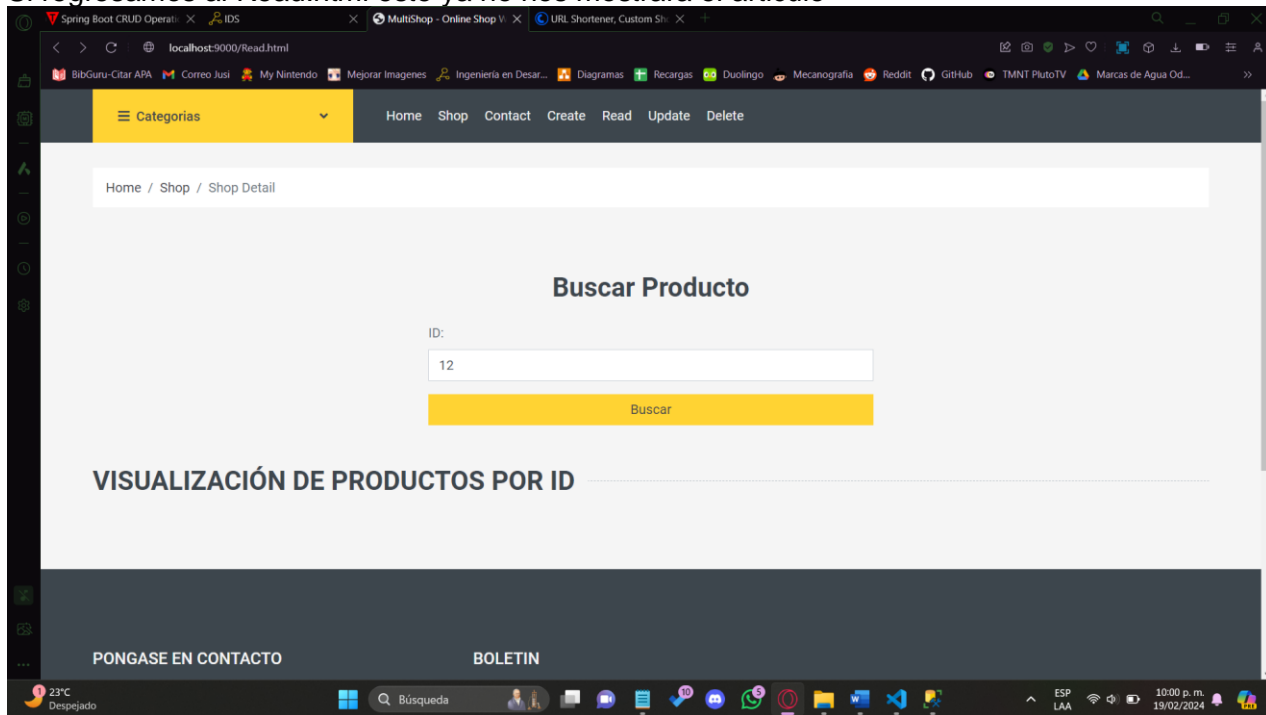
https://cutt.ly/KwBsGCEE

Eliminar Producto

23°C Despejado

Búsqueda

Si regresamos al Read.html este ya no nos mostrara el articulo



Conclusión

La implementación de un API REST para gestionar el carrito de compras en el sitio web de la tienda Sara es fundamental para mejorar la experiencia del usuario y garantizar un proceso de compra eficiente y seguro. Este tipo de funcionalidad no solo es crucial en el ámbito del comercio electrónico, sino que también puede aplicarse en una amplia gama de campos laborales y situaciones de la vida cotidiana.

En el contexto laboral, el desarrollo de un API REST para el carrito de compras permite a las empresas automatizar y simplificar el proceso de gestión de inventario, ventas y pedidos. Además, facilita la integración con otros sistemas y plataformas, lo que brinda una mayor flexibilidad y escalabilidad a la infraestructura tecnológica de la empresa.

En la vida cotidiana, esta funcionalidad puede ser aplicada en diversas situaciones, como la gestión de listas de compras en aplicaciones móviles, la reserva de citas o servicios en línea, e incluso la planificación y coordinación de eventos. La capacidad de agregar, modificar, eliminar y consultar productos de manera eficiente y precisa mejora la productividad y la organización en diferentes aspectos de la vida diaria.

GitHub

<https://github.com/JusiLinGu>