SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE MEXICALI

Alumno:

Lora Jalomo Daniel Alejo

Número de Control:

21490081

Carrera:

Ingeniería en Sistemas Computacionales

Curso:

Programación Web

Docente:

Carlos López Castellanos

Trabajo a presentar:

Reporte de practica Tienda en linea

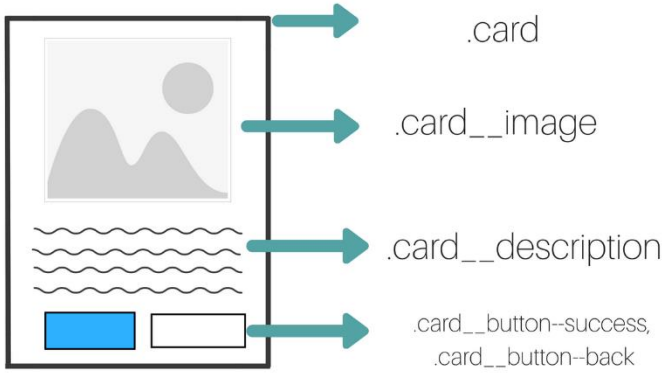
Mexicali, Baja California 28 de marzo del 2024

Enlace a github: https://github.com/Daniellora98/Tienda-virtual-con-JS

Introducción

En este reporte de practica estaremos llevando a cabo los temas fundamentales sobre el desarrollo de la tienda online, utilizando las tecnologías más comunes para el desarrollo web como lo es HTML, CSS y JavaScript, en este primer proyecto un poco mas complejo es donde estaremos utilizando JavaScript un poco mas a fondo, fuera de la teoría y poniendo todo lo aprendido del curso en práctica.

Para esta práctica utilice la metodología **BEM** (Bloques, Elementos, Modificadores). Esto nos ayuda a manejar un mejor gestor de código, más fácil de leer y más fácil de utilizar, manejar mejor las clases CSS en pocas palabras, en el apartado de desarrollo profundizaré más.



Aquí un ejemplo practico para el uso de la metodología BEM, en donde contamos con la clase padre .card, donde podremos ajustar el tamaño global, su margen, cuantas columnas ya sea con flex o grid etc. Y dentro de este contenedor tenemos la imagen y descripción, que actuaran como elementos, y a su vez los botones, que dentro de CSS los manejaremos como modificadores, esto es una pequeña introducción a la metodología BEM.

1. **Explique cada una de las funciones definidas por el usuario en el código JavaScript presentado en el paso 2.**

addEventListener("DOMContentLoaded" Esta función escucha el evento 'DOMContentLoaded', que se dispara cuando el HTML y todos los recursos asociados (como imágenes y estilos) han sido completamente cargados y analizados, sin esperar a que los archivos externos, como hojas de estilo o imágenes, estén descargados. Dentro de esta función, se define la lógica principal del script.

const catalogo = [...]: Aquí se define una matriz llamada 'catalogo' que contiene objetos que representan productos.

catalogo.forEach((producto) => Este método itera sobre cada objeto en la matriz 'catalogo'. Por cada producto en el catálogo, se genera dinámicamente una tarjeta de producto utilizando el contenido HTML dentro de la plantilla de cadena del método 'innerHTML'.

function agregarProductoAlCarrito(producto, cantidad) { Esta es una función que se llama cuando un usuario agrega un producto al carrito. Toma dos parámetros: 'producto' (el objeto del producto seleccionado) y 'cantidad' (la cantidad de ese producto que se agregará al carrito). Esta función actualiza el carrito con el nuevo producto y su cantidad.

function actualizarResumenCompra() Esta función actualiza el resumen de la compra que se muestra en la página. Itera sobre los elementos del carrito, calcula el subtotal para cada producto y muestra el total de la compra

function mostrarTicketCompra() Esta función se encarga de mostrar un ticket de compra en una nueva ventana. Construye el contenido del ticket utilizando los elementos del carrito y luego abre una nueva ventana con este contenido.

2. **En el código js, ¿qué tipo de variable es "catalogo"? ¿Como se manipula?**

Es un array que contiene objetos. Se manipula agregando, eliminando o modificando objetos dentro de él según sea necesario. Por ejemplo, se podría agregar un nuevo producto al catálogo utilizando el método 'push()'.

3. **¿Que hace const card = document.createElement("div") ?**

Esta línea crea un nuevo elemento HTML 'div'. Este elemento se utilizará como contenedor para la tarjeta de producto.

4. **¿Que hace card.innerHTML ?**

Esta propiedad se utiliza para establecer o devolver el contenido HTML de un elemento. En este caso, se utiliza para definir la estructura de la tarjeta de producto, incluyendo la imagen, el título, el precio, el campo de entrada para la cantidad y el botón de "Agregar al Carrito".

5. **¿Que hace catalogoContainer.appendChild(card) ?**

Agrega el elemento 'div' (que representa una tarjeta de producto) como un hijo del elemento con el id 'catalogoContainer'. Esto significa que cada tarjeta de producto generada dinámicamente se agregará al contenedor de catálogo en la página HTML.

Desarrollo

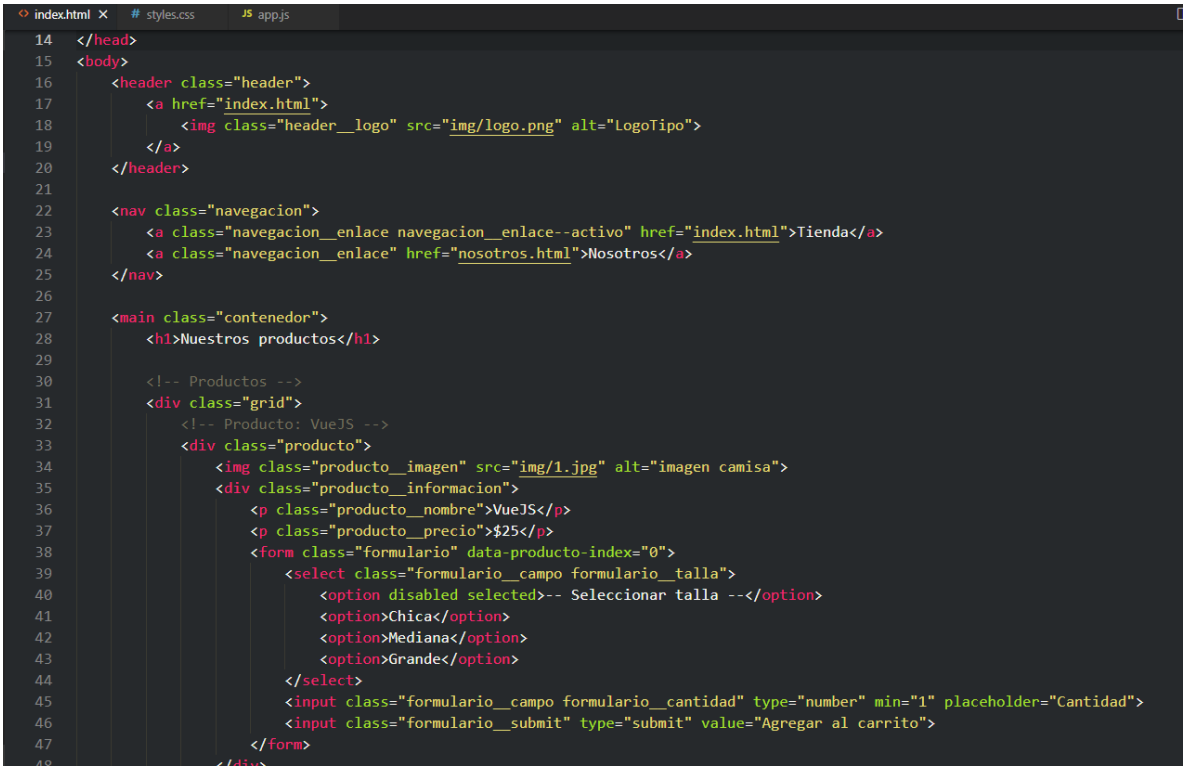
Para esta práctica, utilizaremos 3 archivos, que son index.html, styles.css, app.js

En donde primeramente en este index.html básicamente estará ubicada la estructura de la página principal de la tienda en línea, en donde estarán ubicados 3 productos.

Primeramente, contamos con una etiqueta header, donde contendrá la clase header para darle estilos CSS, dentro de este header esta el logo en este caso de JalomoDev, después de esto contamos con un nav que básicamente será nuestro menú , aquí haciendo uso de BEM antes mencionado solamente colocamos como clase al nav será de navegación por que este será un **bloque,** mientras tanto dentro de este, contamos con dos enlaces, index.html y nosotros.html

Seguido de esto ya contamos con el contenedor principal que tiene la clase contenedor, dentro de este el titulo h1 de nuestros productos.

Utilizamos grid para la colocación de los elementos es por eso que lleva la clase grid, seguido de todos los productos con la antes mencionada metodología BEM, por ejemplo producto__imagen



En esta ultima sección contamos con el resumen de compra, que estaremos utilizando JS , y por ultimo el footer que adopta la clase footer.

```
        <!-- Resumen de la compra -->
        <div id="resumenCompra">
            <h2>Resumen de la compra</h2>
            <ul id="productosSeleccionados"></ul>
            <p id="totalCompra"></p>
        </div>

        <div class="centrarboton">
            <!-- Botón para finalizar la compra -->
            <button id="finalizarCompra">Finalizar compra</button>

        </div>

    </main>

    <footer class="footer">
        <p class="footer__texto">Front End Store - Todos los derechos Reservados 2024.</p>
    </footer>

    <!-- JavaScript -->
    <script src="app.js"></script>
</body>
</html>
```

En esta sección styles.css contamos con las variables globales dentro de :root, que haremos uso de ella para la paleta de colores y para la tipografia de nuestro sitio web, así como un snipet global en la etiqueta html para asignar que 1rem = 10px. Ademas de algunos ajustes al body y a los párrafos.

```
/* Custom properties, variables globales */
:root{
    --primario: #1976D2;
    --primarioOscuro: #004BA0;
    --secundario: #FF6F00;
    --secundarioOscuro: #E65100;
    --blanco: #ffffff;
    --negro: #000;
    /* Custom properties para fuente principal */
    --fuentePrincipal: 'Staatliches', cursive;
}

/* Snipet global  */
html{
    box-sizing: border-box;
    font-size: 62.5%; /* Permite agregar que 1rem = 10px */
}
*, *:before, *:after{
    box-sizing: inherit;
}

/* Globales */
body{
    background-color: var(--primario);
    font-size: 1.6rem;
    line-height: 1.5;
}

p{
    font-size: 1.8rem;
    font-family: Arial, Helvetica, sans-serif;
    color: var(--blanco);
}

```

Continuando con el archivo styles.css a modo de no extender la explicación , solamente nos enfocaremos en la etiqueta navegación, recordemos la metodología BEM ,en este caso el **bloque** se trata que es navegación (nuestro menú), a este le daremos el color de fondo, el padding a la izquierda y derecha, y display flex para el orden de los elementos hijo, asi como el gap para la distancia entre los elementos, por otra parte contamos con al clase navegación__enlace, esta actuara como un **elemento**, le daremos los estilos de fuente principal utilizando los custom properties y las variables, asi como su color y su tamaño de fuente, y por ultimo contamos con el **modificador,** que le indicamos que cuando pasemos el mouse por encima tenga el color de la variable que adopta –secundario. De esta manera completamos los estilos de la navegación utilizando **BEM,** haciendo de nuestro código mucho mas fácil de leer.

```
/* Navegacion */
.navegacion{
    background-color: var(--primarioOscuro);
    padding: 1rem 0;
    display: flex;
    justify-content: center;
    gap: 2rem;
}
.navegacion__enlace{
    font-family: var(--fuentePrincipal);
    color: var(--blanco);
    font-size: 3rem;

}
.navegacion__enlace--activo,
.navegacion__enlace:hover{
    color: var(--secundario);
}
/* Grid */
.grid{
    display: grid;
    grid-template-columns: repeat(2,1fr); /*  2 elementos en dos columnas*/
}
@media (min-width: 768px) { /* Escritorio */
    .grid{
        grid-template-columns: repeat(3, 1fr);
        gap: 2rem;
    }
}
/* Productos */
.producto{
    background-color: var(--primarioOscuro);
    padding: 1rem;
```

Archivo app.js

Como bien sabemos en este archivo es donde estarán las funciones que necesitamos para lograr que la tienda virtual permita la funcionalidad de agregar productos al carrito e ir sacando el total.

Creación de un array de productos: Se define un array llamado productos que contiene objetos que representan productos. Cada objeto tiene dos propiedades: 'nombre' (el nombre del producto) y 'precio' (el precio del producto).

Inicialización de la variable carrito: Se declara una variable llamada carrito, que se utilizará para almacenar los productos seleccionados por el usuario para comprar.

```
document.addEventListener("DOMContentLoaded", function() {
    const productos = [
        { nombre: "VueJS", precio: 25 },
        { nombre: "AngularJS", precio: 25 },
        { nombre: "ReactJS", precio: 25 }
    ];
    let carrito = [];
```

Manejo de la selección de talla y cantidad: Se seleccionan todos los elementos del DOM que tienen la clase 'formulario' y se agrega un event listener para el evento 'submit' a cada uno de ellos. Cuando se envía uno de estos formularios, se evita el comportamiento predeterminado del envío del formulario (que es recargar la página) usando event.preventDefault(). Luego, se obtiene el índice del producto seleccionado, la talla y la cantidad ingresada por el usuario desde los elementos del formulario. Si la talla y la cantidad son válidas se agrega un objeto al carrito que representa el producto seleccionado con su talla y cantidad.

```
    // Manejar la selección de talla y cantidad
    const formularios = document.querySelectorAll(".formulario");
    formularios.forEach(formulario => {
        formulario.addEventListener("submit", function(event) {
            event.preventDefault();
            const productoIndex = parseInt(event.target.dataset.productoIndex);
            const talla = event.target.querySelector(".formulario__talla").value;
            const cantidad = parseInt(event.target.querySelector(".formulario__cantidad").value);
            if (talla && cantidad > 0) {
                carrito.push({ producto: productos[productoIndex], talla, cantidad });
                actualizarResumenCompra();
            } else {
                alert("Por favor selecciona talla y cantidad válidas.");
            }
        });
    });
```

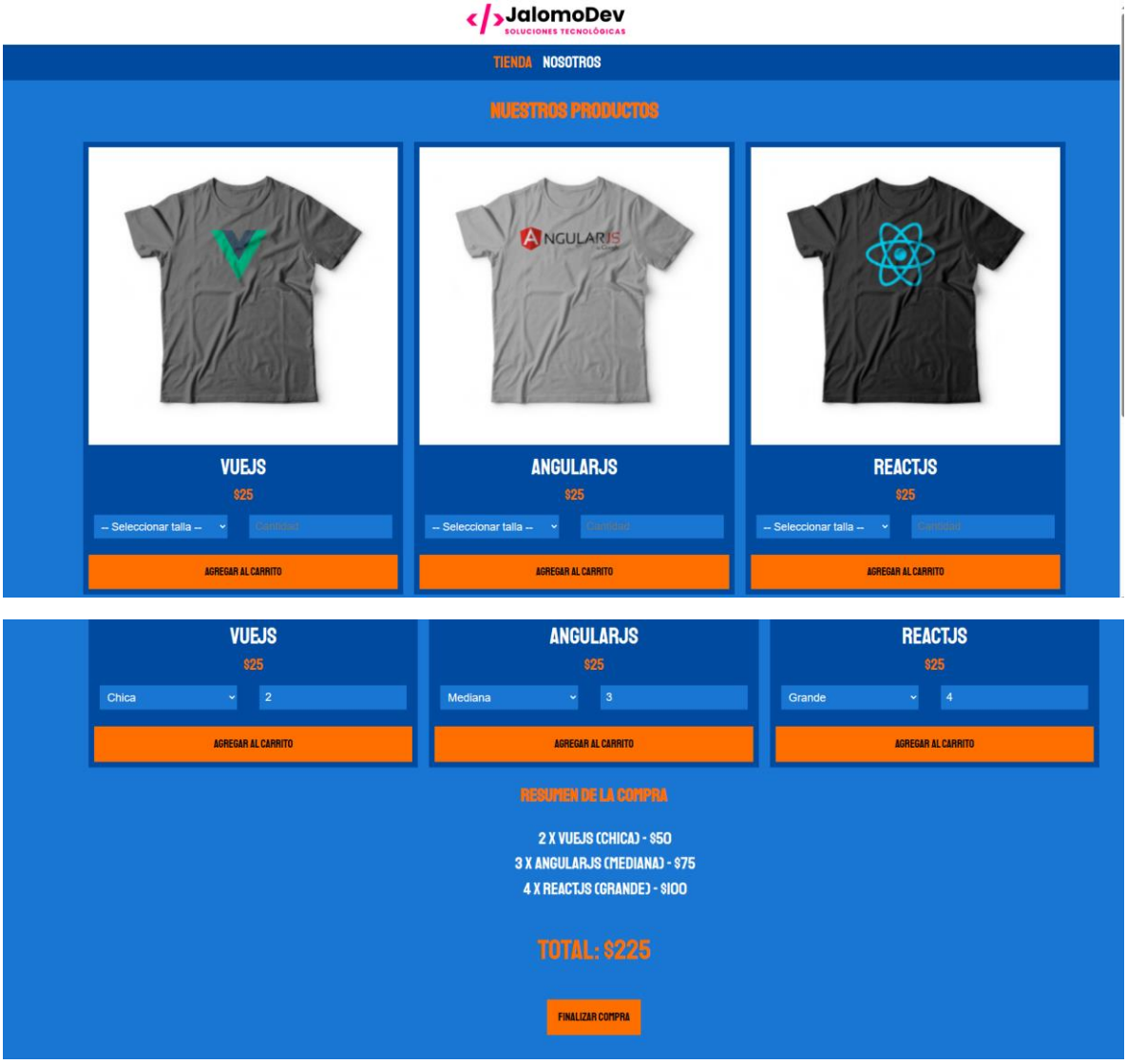
Función actualizarResumenCompra() : Esta función actualiza el resumen de la compra en la página. Primero, limpia el contenido del elemento con el id 'productosSeleccionados'. Luego, itera sobre los productos en el carrito y calcula el subtotal para cada uno de ellos. Se muestra una lista de los productos seleccionados con su cantidad, nombre, talla y subtotal en el elemento 'productosSeleccionados'. También se actualiza el total de la compra en el elemento con el id 'totalCompra'.

```
    // Actualizar el resumen de la compra
    function actualizarResumenCompra() {
        const productosSeleccionadosElement = document.getElementById("productosSeleccionados");
        productosSeleccionadosElement.innerHTML = "";
        let totalCompra = 0;
        carrito.forEach(item => {
            const subtotal = item.producto.precio * item.cantidad;
            totalCompra += subtotal;
            const listItem = document.createElement("li");
            listItem.textContent = `${item.cantidad} x ${item.producto.nombre} (${item.talla}) - $${subtotal}`;
            productosSeleccionadosElement.appendChild(listItem);
        });
        document.getElementById("totalCompra").textContent = `Total: $${totalCompra}`;
    }
```

Mostrar el resumen de la compra al hacer clic en "Finalizar compra": Se agrega un event listener para el evento 'click' al botón con el id 'finalizarCompra'. Cuando se hace clic en este botón, se llama a la función actualizarResumenCompra() para actualizar el resumen de la compra en la página. Luego, se muestra el elemento con el id 'resumenCompra' estableciendo su estilo 'display' en 'block', lo que lo hace visible para el usuario. Finalmente, se abre una nueva ventana con un ticket que muestra los productos seleccionados, sus cantidades, tallas y subtotales.

```
    // Mostrar el resumen de la compra al hacer clic en "Finalizar compra"
    document.getElementById("finalizarCompra").addEventListener("click", function() {
        actualizarResumenCompra();
        document.getElementById("resumenCompra").style.display = "block";
        // Abrir una nueva ventana con el ticket de los productos obtenidos
        const ticket = carrito.map(item => `${item.cantidad} x ${item.producto.nombre} (${item.talla}) - $${item.producto.precio *
        const nuevaVentana = window.open("", "_blank");
        nuevaVentana.document.write("<pre>" + ticket + "</pre>");
    });
});
```

Resultados



Conclusión

Esta practica refuerza nuestro aprendizaje adquirido en las primeras unidades, tanto refuerza nuestro conocimiento sobre las etiquetas HTML , tanto nuestro conocimiento en CSS, como centrar los elementos, como crearlos, modificarlos y más que nada en esta práctica añadí la metodología BEM, es decir voy añadiendo mejores practicas de diseño web a medida que se avanza, y sobre todo y lo más importante el manejo de JavaScript que eso es lo fundamental de esta práctica, crear la tienda online y permitir agregar productos nos seria imposible sin JavaScript, por que para esto necesitamos un lenguaje de programación, hemos aprendido bastante en esta práctica, solidificando nuestros conceptos teóricos en prácticos.