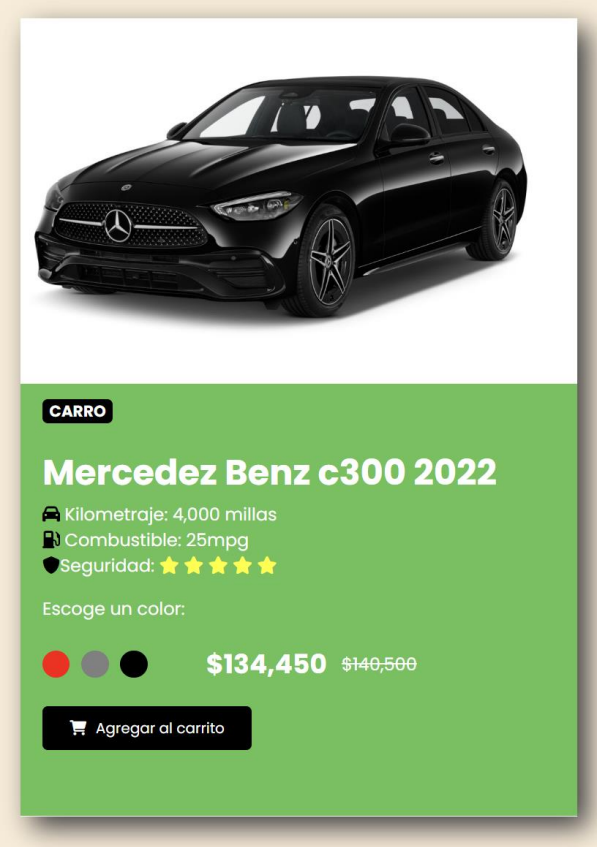

IMPORTANTE: Tenéis todo el código del proyecto en un archivo comprimido para ir analizándolo conforme se va leyendo este documento para poder entender todo lo que hace esta aplicación.

Introducción

Si estuvieras escuchando música en una aplicación y quisieras pausar o pasar una canción, tendrías que hacerlo a través de la aplicación.

Este proceso es similar a cómo funciona el Modelo de objetos de documento o DOM. Aquí, la aplicación de música representa el DOM porque sirve como medio para realizar cambios en la música.

Vamos a aprender qué es el DOM y cómo funciona de una manera práctica.

¿Qué es el DOM?

El DOM es una API Web que permite a los desarrolladores usar la lógica de programación para realizar cambios en el código HTML. Es una manera confiable de realizar cambios que convierten los sitios web estáticos en dinámicos.

Este es un tema importante en el desarrollo web porque el DOM sirve para dar inicio al uso de JavaScript en el navegador.

El código HTML no es considerado parte del DOM hasta que es analizado por el navegador.

Para ver que está pasando en el código HTML, cuando está siendo analizado, copia tu código desde la etiqueta <body> y pégalo aquí (dentro del cuadro con el título 'Markup to test' después de los tres puntos).

Funcionalidades del Proyecto

Como puedes ver arriba en la demostración del proyecto, estas son las funcionalidades que estaremos implementado:

1. **Cambio dinámico de color**: Cuando se hace clic en un color, el color del carro, el botón addTocart, y la etiqueta, todos cambian para coincidir con el color seleccionado.
2. **Botón de cambio**: Al hacer clic en el botón addToCart aparece el botón de éxito y viceversa.

Código HTML:

En nuestro archivo index.html, vamos a crear la estructura básica del proyecto, lo cual incluye: el enlace del archivo CSS, Font Awesome, y Google Fonts – todo dentro de nuestra etiqueta <head>. Dentro de nuestra etiqueta <body>, crearemos nuestra tarjeta del producto y enlazaremos nuestra etiqueta JavaScript al final de la etiqueta <body>.

Código CSS:

En nuestro archivo de estilos, style.css, vamos a configurar nuestros estilos generales. Y damos estilos a nuestro producto, empezando con la etiqueta, imagen, descripción y detalles.

También le daremos estilos a nuestros colores: sus precios, colores como un grupo, y colores individuales.

Finalmente, le daremos estilos a nuestros botones.

Implementación del DOM

Todo en el DOM recae en alguna de estas dos categorías: selección de elementos y manipulación de elementos. Después de crear nuestros archivos HTML y CSS, nos dirigimos a nuestro archivo app.js para implementar lo siguiente:

1. **Selección**: Hacemos referencia a todos los elementos que queremos hacer dinámicos desde nuestro código HTML y les asignamos variables en nuestro archivo JavaScript.
2. **Manipulación**: Una vez que hemos seleccionado y enlazado las variables, creamos las diversas funciones responsables de la manipulación y luego las enlazamos a las variables.

¿Cómo seleccionar elementos en el DOM?

Para tener acceso a los elementos de HTML que se quieren manipular, necesitas hacer que JavaScript sea consciente de la existencia de dichos elementos. A esto nos referimos generalmente como "selección" de elementos – básicamente es enlazarlos.

En el DOM, no hay una sola forma de localizar y hacer referencia a un elemento para su manipulación. En cambio, dependerá del selector que hayas usado en la etiqueta del elemento.

Para ello, se asigna el elemento a una variable. Tomando el siguiente formato. Ten en cuenta que todos los selectores del DOM están precedidos por el objeto de documento y un punto:

```
const ejemplo = document.[DOMselector]
```

En nuestro archivo de JavaScript, tenemos que seleccionar todos los elementos que queremos manipular, como el botón, los colores, la tarjeta de imagen, y la etiqueta.

Vamos a utilizar tantos selectores de DOM como sea posible, así que aprendamos más sobre ellos.

Cómo usar el **`querySelector`**

querySelector es un método que acepta el selector exacto del CSS en una cadena y retorna un elemento. Puedes usarlo para seleccionar los colores rojo y negro así como la tarjeta de imagen, usando sus nombres de clase.

Si quisieras usar este enfoque para seleccionar y retornar más de un elemento, puedes usar QuerySelectorAll en su lugar.

```
const redColor = document.querySelector(".red");
```

El código de arriba vincula el span con la clase "red" `<span class="red"></span>` de nuestro código HTML a la variable redColor en nuestro JavaScript.

```
const blackColor = document.querySelector(".black");
```

El código de arriba vincula el span con la clase "black" `<span class="black"></span>` de nuestro código HTML a la variable blackColor en nuestro JavaScript.

```
const imageCard = document.querySelector(".product-image");
```

El código de arriba vincula el div con la clase "product-image" `<div class="product-image">` de nuestro código HTML a la variable imageCard en nuestro JavaScript.

```
const feedbackBtn = document.querySelector(".feedback");
```

El código de arriba vincula el botón con la clase "feedback" `<button class="feedback">` de nuestro código HTML a la variable feedbackBtn en nuestro JavaScript.

Cómo usar getElementsByClassName

Puedes usar este selector para seleccionar el color gris. Es muy similar al `querySelector`. La única diferencia es que este método acepta solo el nombre de la clase, sin el punto anterior (.)

```
```

```
const grayColor = document.getElementsByClassName("gray");
```

El código de arriba vincula el span con la clase "gray" `<span class="gray"></span>` de nuestro código HTML a la variable grayColor en nuestro JavaScript.

Cómo usar getElementById

Puedes usar este selector para seleccionar el botón del carrito. Es muy similar al `getElementsByClassName`. La única diferencia es que debido a que usamos el ID para mostrar que es único, se usa solo en un elemento. Este método lee `getElement`, sin s al final.

```
const cartButton = document.getElementById("button");
```

El código de arriba vincula el botón con el id "button" `<button id="button">` de nuestro código HTML a la variable cartButton en nuestro JavaScript.

Cómo usar GetElementsByTagName

Los atributos no son la única forma de seleccionar un elemento. También puedes usar nombres de etiqueta. Si has utilizado la etiqueta a la que haces referencia más de una vez, entonces retornara una lista de elementos. Use la indexación para seleccionar la correcta.

```
const itemTag = document.getElementsByTagName("h3")[0];
```

El código de arriba enlaza el h3 que contiene nuestra etiqueta de producto `<h3 class="tag">` de nuestro código HTML con la variable itemTag en nuestro JavaScript.

De todos estos, el `querySelector` y `querySelectorAll` son probablemente los más populares debido a como generalizan y lo menos restrictivos que son.

Cómo manipular elementos en el DOM

La manipulación es el objetivo principal del DOM. Es todo lo que sucede después de hacer referencia y seleccionar los elementos con los que se desea trabajar. Esto conduce a un cambio en el estado del elemento, de estático a dinámico.

Dos conceptos que debe conocer para entender la manipulación del DOM son los **eventos** y los **manejadores**.

¿Qué son los Eventos?

Vamos a usar la misma analogía de música del inicio. En la aplicación de música, debes realizar una acción [hacer clic o deslizar] antes de que se activen las funcionalidades.

En el DOM, esta acción es conocida como un evento. Tenemos muchos eventos como el clic, desplazar, pasar el cursor por encima, cambiar, y muchos más.

En el DOM, las respuestas están sujetas a cada evento. Es decir, que debe suceder un evento para obtener una respuesta. Esto es conocido como un event listener. El detector de eventos usualmente viene en forma de un método `addEventListener` y este toma dos argumentos (evento, y manejador de evento).

Anatomía de un evento

Los eventos del DOM normalmente contienen un elemento, su event listener, y una función.

```
element.[eventListenerMethod(event, eventHandler)
```

¿Qué son los manejadores de Eventos?

Los manejadores de eventos son las respuestas enviadas cuando nuestro método event listener lee un evento. Sin un manejador de eventos, no tenemos manera de alertar a nuestro código de que un evento ha ocurrido.

Todas las modificaciones que suceden en el DOM, tales como estilizar, agregar, eliminar, entre otros, son responsabilidad de los manejadores de eventos. Estas son las funciones que se

encuentran en el segundo argumento del método addEventListener. Estas están siempre alerta para ejecutarse tan pronto ocurra el evento (el primer argumento).

```
redColor.addEventListener("click", function () {
    cartButton.style.backgroundColor = "red";
    itemTag.style.backgroundColor = "red";
    imageCard.style.backgroundImage = 'url("./img/red-benz.webp")';
  });
```

En el código de arriba, la función después del evento 'clic' es el manejador de evento. Esto quiere decir que todo lo que está dentro de la función será implementado cuando al color rojo se le haya hecho clic.

¿Cómo implementar los Eventos y los manejadores de Eventos?

En este proyecto, vamos a usar eventos y manejadores de eventos en 5 implementaciones. Veremos cada una de ellas a continuación.

Primero, los usaremos para **hacer el color rojo funcional**. Una vez que el usuario haga clic en el color rojo, al botón del carrito, y a la etiqueta del elemento se le agregará un fondo de color rojo en los estilos. La imagen del carro también cambiará a un color rojo.

Hacemos esto tomando la variable redColor y agregando un event listener, de 'clic'. Esto quiere decir, que cuando se haga clic en el color rojo, queremos que nuestro código sea alertado. En respuesta, la function manejadora del evento es ejecutada inmediatamente.

```
redColor.addEventListener("click", function () {
    cartButton.style.backgroundColor = "red";
    itemTag.style.backgroundColor = "red";
    imageCard.style.backgroundImage = 'url("./img/red-benz.webp")';
  });
```

Ahora, haremos **el color gris funcional**. Cuando un usuario haga clic en el color gris, al botón del carrito, y a la etiqueta del elemento se le agregará un fondo de color gris en los estilos. La imagen del carro también cambiará a un color gris.

Hacemos esto tomando la variable grayColor y agregando un event listener de 'clic'. Esto quiere decir, que cuando se haga clic en el color gris, queremos que nuestro código sea alertado. En respuesta, la function manejadora del evento es ejecutada inmediatamente.

```
grayColor[0].addEventListener("click", function () {
    cartButton.style.backgroundColor = "gray";
    itemTag.style.backgroundColor = "gray";
    imageCard.style.backgroundImage = 'url("./img/gray-benz.jpg")';
  });
```

Siguiendo el mismo patrón, haremos **el color negro funcional**. Cuando un usuario haga clic en el color negro, al botón del carrito, y a la etiqueta del elemento se le agregará un fondo de color negro en los estilos. La imagen del carro también cambiará a un color negro.

Hacemos esto tomando la variable blackColor y agregando un event listener de 'clic'. Esto quiere decir, que cuando se haga clic en el color negro, queremos que nuestro código sea alertado. En respuesta, la function manejadora del evento es ejecutada inmediatamente.

```
blackColor.addEventListener("click", function () {
    cartButton.style.backgroundColor = "black";
    itemTag.style.backgroundColor = "black";
    imageCard.style.backgroundImage = 'url("./img/black-benz.jpg")';
  });
```

Hemos visto un enfoque para los manejadores de eventos, el cual consiste en crear la función dentro del método addEventListener.

Otro enfoque es crear la función antes de pasar el nombre de la función como un argumento en el método addEventListener.

Cómo implementar el botón del Carrito

Empezaremos creando una funcionan llamada cart. La función cart esconde el botón del carrito y muestra el botón de retroalimentación. El nombre de la función cart es entonces pasado al método event listener, como segundo argumento.

```
const cart = () => {
    cartButton.style.display = "none";
    feedbackBtn.style.display = "block";
  };
  cartButton.addEventListener("click", cart);
```

Cómo Implementar el botón de retroalimentación

Primero creamos una función llamada feedback. La función feedback esconde el botón de retroalimentación y muestra el botón del carrito. El nombre de la función feedback es entonces pasado al método event listener como un segundo argumento.

```
const feedback = () => {
    cartButton.style.display = "block";
    feedbackBtn.style.display = "none";
  };
  feedbackBtn.addEventListener("click", feedback);
```

Conclusión

El DOM es una parte esencial del desarrollo web moderno porque este ayuda a los desarrolladores a transformar aplicaciones de páginas web estáticas a dinámicas.

Como principiantes, puede ser difícil entender el DOM y todo lo que conlleva su manejo. Tomar tiempo para construir proyectos simples como este te ayudará a reforzar los conceptos.