| ESPE UNIVERSIDAD DE LAS FUERZAS ARMADAS INNOVACIÓN PARA LA EXCELENCIA | INFORME DE PRÁCTICA DE LABORATORIO |
|---|---|

| | | | | | |
|---|---|---|---|---|---|
| **DEPARTAMENTO:** | **CIENCIAS DE LA COMPUTACIÓN** | **CARRERA:** | **INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN** | | |
| **ASIGNATURA:** | Programación Integrativa de Componentes | **NIVEL:** | 6 | **FECHA:** | 23/05/25 |
| **DOCENTE:** | Ing. Paulo Galarza | **PRÁCTICA N°:** | 1 | **CALIFICACIÓN:** | |

# Desarrollar una Tarjeta de Producto para un E-commerce con Composición de Componentes

**Anderson Arquimedes Campos Alvarado**


## RESUMEN

El proyecto consistió en implementar un Custom Element que encapsula la estructura y estilos usando Shadow DOM, soporta atributos dinámicos (img, title, price, description) y permite actualizarse mediante cambios en dichos atributos. Para lograrlo, se creó una rama específica donde se corrigieron problemas técnicos relacionados con un loop infinito en el manejo de atributos. Se realizó una prueba automatizada con JavaScript para validar la actualización dinámica del componente sin necesidad de recargar la página.


**Palabras clave:** Web Components, Shadow DOM, Reactividad

## 1. INTRODUCCIÓN:

Se desarrolló un componente web personalizado utilizando tecnologías modernas como Web Components y Shadow DOM. El objetivo principal fue crear un <product-card> que permitiera mostrar información dinámica de productos con encapsulación de estilos y funcionalidad. Además, se abordó la integración de atributos reactivos para actualizar el contenido en tiempo real y la solución de errores técnicos surgidos durante el desarrollo.

## 2. OBJETIVO(S):

### 2.1 Objetivo general

- Implementar un componente web personalizado que permita mostrar información dinámica de productos con encapsulación de estilos y funcionalidad, mejorando la modularidad y reactividad en aplicaciones web.

### 2.2. Objetivos específicos

- Crear un Custom Element con Shadow DOM que encapsule estilos y estructura, garantizando su independencia visual y funcional.

- Implementar la actualización dinámica de datos mediante atributos observados, permitiendo la manipulación de contenido en tiempo real.
- Gestionar el desarrollo mediante control de versiones con Git, utilizando ramas para aislar cambios y facilitar la integración ordenada de mejoras.

## 3. MARCO TEÓRICO

Los Web Components representan un conjunto de estándares web que permiten la creación de elementos HTML personalizados y reutilizables con encapsulación total de estructura, estilo y comportamiento (W3C, 2023).

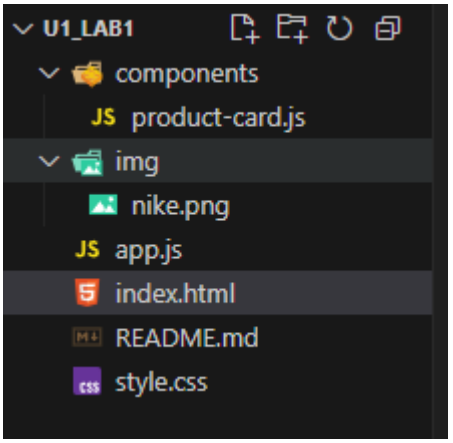
Una pieza fundamental de los Web Components es el Shadow DOM, que proporciona un árbol DOM encapsulado y aislado dentro de un componente, garantizando que los estilos y scripts del componente no afecten ni sean afectados por el DOM global de la página (Mozilla Developer Network, 2024).

El manejo de atributos a través del método attributeChangedCallback permite que los componentes reaccionen dinámicamente a cambios en sus propiedades externas, actualizando su interfaz de forma reactiva y mejorando la experiencia de usuario (Google Developers, 2023). Asimismo, el uso de templates y slots permite la composición flexible, otorgando al desarrollador la capacidad de insertar contenido dinámico o personalizado dentro del componente.

Por último, la implementación de variables CSS (CSS Custom Properties) dentro del Shadow DOM posibilita la personalización visual sin romper la encapsulación, facilitando la reutilización y adaptación de componentes a diferentes estilos o temáticas (W3Schools, 2024).

## 4. DESARROLLO

Primero desarrollaremos una estructura para nuestro laboratorio, salen cambios en el git porque se cambió el nombre de las carpetas por los cambios de las rutas.



### Paso 1: Configuración del Proyecto

Se crea una estructura base del proyecto con los archivos necesarios y se definió el uso inicial del componente <product-card> en index.html.

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>Proyecto</title>
  <link rel="stylesheet" href="style.css"/>
</head>
<body>
  <product-card
    img="./img/nike.png"
    title="MOOGLE DAYDREAM TM"
    price="$599"
    description="Lorem ipsum dolor consectetur">
  </product-card>

  <script type="module" src="app.js"></script>
</body>
</html>
```

### Paso 2: Creación del Custom Element

Se desarrolla la clase productCard extendiendo HTMLElement. Se implementó el Shadow DOM para encapsular estilos y estructura, y se registró el elemento con customElements.define.

```js
class productCard extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: "open" });
  }

  static get observedAttributes() {
    return ["img", "title", "price", "description"];
  }

  attributeChangedCallback(name, oldValue, newValue) {···
  }

  get template() {···
  }

  render() {···
  }

  connectedCallback() {···
  }
}
customElements.define("product-card", productCard);
```

### Paso 3: Integración de Datos Dinámicos

Se definieron los atributos observados (img, title, price, description) y se implementó attributeChangedCallback para actualizar la interfaz cuando cambian dichos atributos.

```
class productCard extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: "open" });
  }

  static get observedAttributes() {
    return ["img", "title", "price", "description"];
  }

  attributeChangedCallback(name, oldValue, newValue) {
    this[name] = newValue;
    this.render();
  }
```

**Paso 3.1. Ingreso de Datos por consola**

attributeChangedCallback en el componente esta función está definida dentro de la clase productCard y es la que se ejecuta automáticamente cuando cambian los atributos que el componente observa

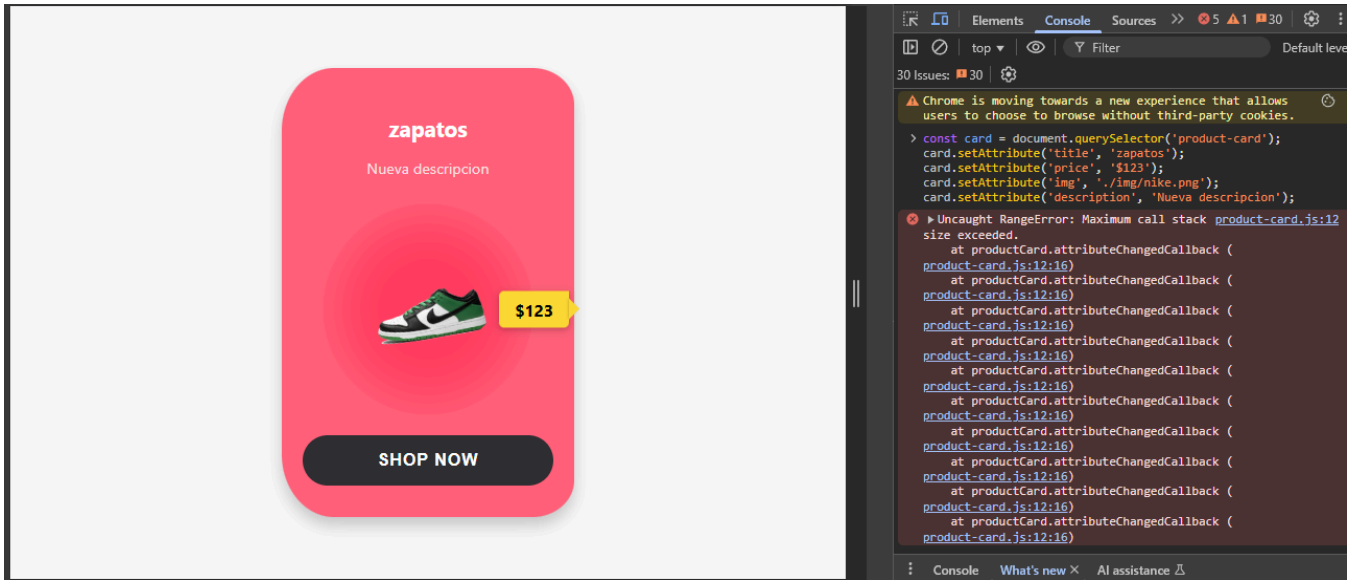
```
  static get observedAttributes() {
    return ["img", "title", "price", "description"];
  }

  attributeChangedCallback(name, oldValue, newValue) {
    this[name] = newValue;
    this.render();
  }
```

Aquí antes del contenedor para los estilos vamos a introducir una funcion render esto ayudara a solucionar el error que muestra la ejecucion en la imagen del paso en el git compartido no esta esta funcion esa es una de las razones por las que se creo la rama al no estar en el tiempo de dos horas estimuladas por el ingeniero.

```
attributeChangedCallback(name, oldValue, newValue) {
  this[name] = newValue;
  this.render();
}
funcion render() {}
get template() {
  const template = document.createElement("template");
  template.innerHTML = `
    <style>
      :host {
```

Ejecución antes de los cambios de la rama




**¿Por qué me da el error?**

En la versión que tenías antes, en el método attributeChangedCallback hacías esto

```
attributeChangedCallback(name, oldValue, newValue) {
  this[name] = newValue; este newValues puede disparar de nuevo el atributo generando un recursividad infinita
  this.render();
}
```

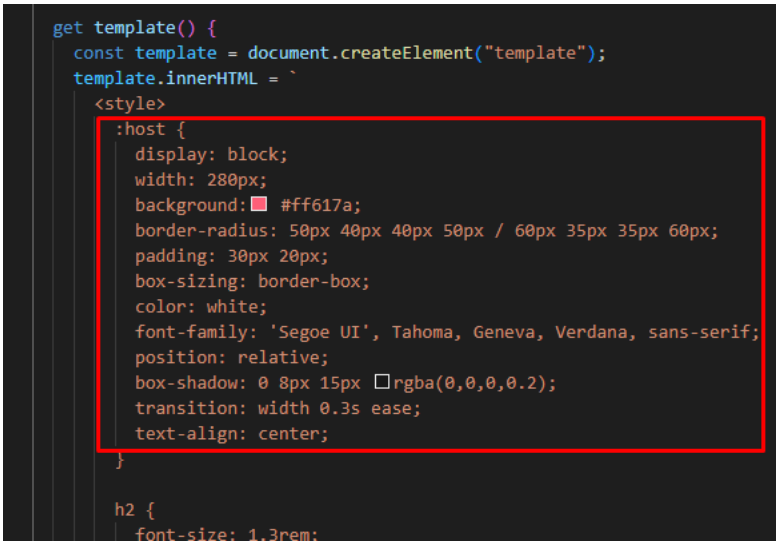
Para solucionar ya no se le asignó valores directos en lugar de eso se obtienen directamente de los atributos con getAttribute()

```
get template() {
  // Tomamos valores directamente desde los atributos
  const title = this.getAttribute('title') || 'Título del producto';
  const price = this.getAttribute('price') || '$0';
  const img = this.getAttribute('img') || './img/nike.png';
  const description = this.getAttribute('description') || 'Descripción de
```

Esto significa que el componente lee el estado actual de sus atributos solo cuando renderiza, sin modificar propiedades internas que disparen callbacks.

**Paso 4: Estilos personalizados con CSS Variables**

Se encapsularon los estilos CSS dentro del Shadow DOM, aplicando variables CSS para facilitar futuras personalizaciones.

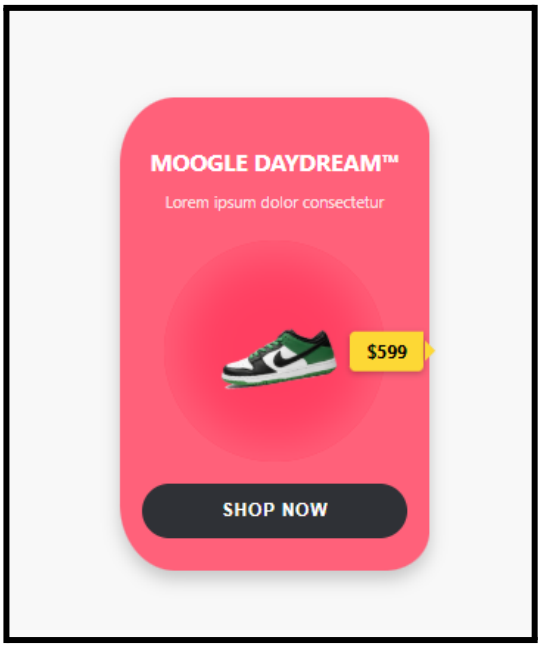



**Paso 5: Pruebas y Documentación**

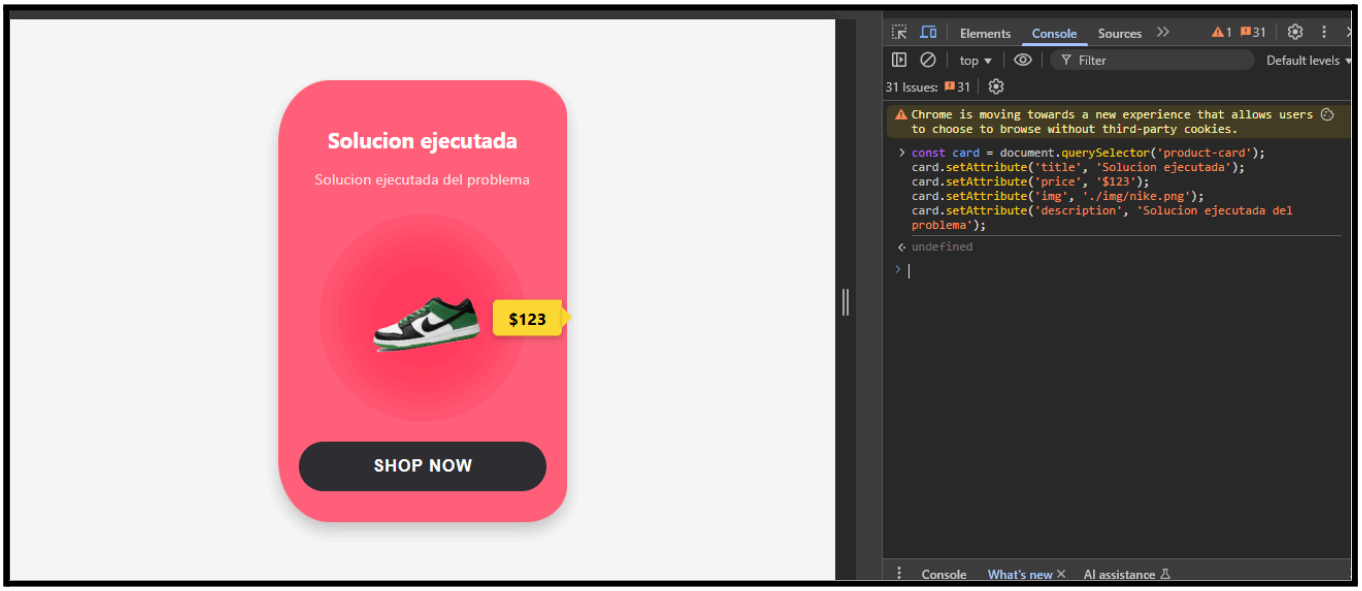
Se verificó la funcionalidad del componente, comprobando que las actualizaciones dinámicas de atributos reflejan los cambios en la tarjeta. Se realizó una prueba automatizada en app.js para modificar atributos tras la carga.


Este código en app.js no es estrictamente necesario para que el laboratorio cumpla con los requisitos mínimos, pero se le agrego porque sirve para demostrar o probar la actualización dinámica de los atributos en tiempo real, después de que la página carga y sirve como una guia para probar la reactividad sin tener que recargar la página o modificar manualmente el HTML.

Ejecución sin la función de manipulación dinámica de atributos o actualización reactiva de atributos en un Custom Element.




Hay que recalcar que se creó una rama para ingresar esa parte antes de las modificaciones antes de la creación de la rama el código si permite el ingreso de datos dinámicos sólo se le hizo el commit para cambiar el nombre de carpetas y eliminar elementos que no servían ya que no se ejecutaba correctamente el proyecto por conflicto de rutas y para solucionar el error del paso 3 ya arriba se explica el error este es la ejecución sin errores.



## 5. CONCLUSIONES

- La utilización de Web Components y Shadow DOM aportó ventajas significativas en modularidad, encapsulación y reutilización, esenciales para proyectos escalables y mantenibles.
- Finalmente, la implementación de la actualización reactiva mediante atributos dinámicos mejora la experiencia del usuario y la flexibilidad del componente en distintas aplicaciones.
- La creación de la rama Lab1_Cambios permitió aislar y solucionar el error de loop infinito generado por la asignación directa de atributos, facilitando un desarrollo más organizado y controlado.

## 6. RECOMENDACIONES

- Mantén la práctica de utilizar ramas en Git para separar funcionalidades y correcciones, facilitando la revisión y evitando errores en la rama principal.
- Explorar la ampliación del componente con más atributos o funcionalidades, como eventos personalizados para interacción con el usuario (ejemplo: añadir al carrito).
- Considerar la optimización del rendimiento al manejar múltiples instancias del componente en páginas con alto volumen de datos.
- No olvides documentar detalladamente cada componente y su API (atributos, métodos, eventos) para facilitar su uso y mantenimiento en proyectos futuros.

## 7. BIBLIOGRAFÍA:

Google Developers. (2023). Using Web Components. Recuperado de https://developers.google.com/web/fundamentals/web-components

Mozilla Developer Network (MDN). (2024). Shadow DOM. Recuperado de https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_shadow_DOM

W3C. (2023). Web Components. Recuperado de https://www.w3.org/TR/components-intro/

W3Schools. (2024). CSS Variables. Recuperado de https://www.w3schools.com/css/css3_variables.asp