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

DEPARTAMENTO:	Ciencias de la Computación - DCCO-SS	**CARRERA:**	Ingeniería en tecnologías de la Información		
ASIGNATURA:	Programación Integrativa	**NIVEL:**	Sexto	**FECHA:**	17/05/2025
DOCENTE:	Ing. Paulo Galarza	**PRÁCTICA N°:**	2	**CALIFICACIÓN:**	

Integrar Elementos HTML en un Web Component

ANTHONY GEOVANNY MEJIA GAIBOR

RESUMEN

En esta práctica de laboratorio, se desarrolló un componente web personalizado denominado <formulario-actualizar>, diseñado para interactuar con el componente <tarjeta-usuario> creado en la tarea anterior. Este nuevo componente permite a los usuarios actualizar el nombre mostrado en la tarjeta de usuario a través de un formulario que incluye validación de entrada. El propósito principal de esta tarea fue aplicar conceptos avanzados de Web Components, como el uso de Shadow DOM para encapsular estilos y estructura, y slots para permitir la personalización del contenido. Además, se buscó fortalecer las habilidades en la gestión de proyectos con GitHub, incluyendo la creación de ramas específicas y la documentación adecuada del trabajo realizado. A lo largo del desarrollo, se resolvieron desafíos relacionados con la sincronización del DOM mediante el evento DOMContentLoaded. Este proceso destacó la importancia de comprender la interacción entre el Shadow DOM y el light DOM, así como la utilidad de las validaciones para mejorar la usabilidad. En conclusión, esta práctica consolidó el conocimiento en la creación de interfaces modulares y reutilizables, enfatizando las buenas prácticas de desarrollo.

Palabras Clave: Web Components, Shadow DOM, slots.

1. INTRODUCCIÓN:

En el ámbito del desarrollo web moderno, los Web Components han emergido como una herramienta fundamental para crear interfaces reutilizables y modulares. Estos componentes permiten a los desarrolladores encapsular la lógica, el estilo y la estructura de elementos de la interfaz, facilitando así la mantenibilidad y escalabilidad de las aplicaciones web. En este contexto, la tarea 2 del laboratorio de programación integrativa se enfocó en la creación de un componente web personalizado, <formulario-actualizar>, diseñado para interactuar dinámicamente con el componente <tarjeta-usuario> desarrollado previamente. Este nuevo componente no solo permite a los usuarios actualizar el nombre mostrado en la tarjeta de usuario a través de un formulario con validación, sino que también sirve como un ejercicio práctico para profundizar en el uso de tecnologías avanzadas como el Shadow DOM y los slots.

La realización de estas actividades se llevó a cabo con un enfoque disciplinado y metódico. Se utilizó GitHub para gestionar el repositorio del proyecto, creando una rama específica (tarea2-integracion-html) para mantener el trabajo de la tarea 2 separado del código base original. Esta práctica no solo facilitó la organización del código, sino que también permitió un seguimiento claro de los cambios y mejoras implementadas. Además, se prestó especial atención a la documentación del proyecto, asegurando que cada paso del desarrollo estuviera bien registrado y que las capturas de pantalla en la carpeta docs/ evidenciaran la interacción entre los componentes.

Durante el desarrollo, se enfrentaron desafíos técnicos significativos, como problemas de sincronización del DOM que impedían la correcta actualización del nombre en el componente <tarjeta-usuario>. Estos desafíos fueron superados mediante la implementación del evento DOMContentLoaded, que aseguró que el DOM estuviera completamente cargado antes de intentar modificar los elementos. Este enfoque no solo resolvió el problema inmediato, sino que

también reforzó la importancia de comprender la interacción entre el Shadow DOM y el light DOM en los Web Components.

2. OBJETIVO(S):

2.1 Desarrollar un componente web que utilice elementos HTML nativos dentro del Shadow DOM y slots para la personalización del contenido.

2.2 Establecer una comunicación efectiva entre el nuevo componente y el componente existente, permitiendo la modificación dinámica de datos.

2.3 Aplicar buenas prácticas de gestión de proyectos, incluyendo el uso de control de versiones con GitHub y la documentación adecuada del trabajo realizado.

3. DESARROLLO

ConnectedCallback

El código proporcionado corresponde al método connectedCallback de un componente web personalizado, escrito en JavaScript para un elemento HTML personalizado. Este método se ejecuta cuando el componente es insertado en el documento, permitiendo que su lógica interna se active en el momento adecuado.

El propósito principal de este código es configurar una funcionalidad que permita al usuario interactuar con un botón y, a partir de esa interacción, validar un campo de entrada para luego actualizar el contenido de otro componente web en la página.

Código

```
connectedCallback() {
        document.addEventListener('DOMContentLoaded', () => {
            const slotBoton =
this.shadowRoot.querySelector('slot[name="boton-enviar"]');
            const botonAsignado = slotBoton.assignedElements()[0] ||
this.shadowRoot.querySelector('slot[name="boton-enviar"] button');
```

```
            if (botonAsignado) {
                botonAsignado.addEventListener('click', () => {
                    const nombreIngresado =
this.shadowRoot.querySelector('#nombreNuevo').value.trim();
                    // Validación simple: no vacío y al menos 3
caracteres
                    if (nombreIngresado === "") {
                        console.log("Error: El nombre no puede estar
vacío.");
                        alert("Por favor, ingresa un nombre.");
                        return;
                    }
                    if (nombreIngresado.length < 3) {
                        console.log("Error: El nombre debe tener al
menos 3 caracteres.");
                        alert("El nombre debe tener al menos 3
caracteres.");
                        return;
                    }

                    const tarjetaUsuario =
document.querySelector('tarjeta-usuario');
                    if (tarjetaUsuario) {
                        const slotNombre =
tarjetaUsuario.shadowRoot.querySelector('slot[name="nombre"]');
                        if (slotNombre) {
                            const elementoNombre =
slotNombre.assignedElements()[0];
                            if (elementoNombre) {
                                elementoNombre.textContent =
nombreIngresado;
                                console.log("¡Nombre cambiado a " +
nombreIngresado + "!");
                            } else {
                                console.log("Error: No se encontró el
elemento asignado al slot 'nombre'.");
                            }
                        } else {
                            console.log("Error: No se encontró el slot
'nombre' en tarjeta-usuario.");
                        }
```

```
                    } else {
                        console.log("Error: No se encontró el componente
tarjeta-usuario.");
                    }
                });
            } else {
                console.log("No se encontró el botón en el slot
'boton-enviar'.");
            }
        });
    }
}
```

Funcionamiento General

Espera a que el DOM esté listo:

- El código registra un listener para el evento DOMContentLoaded, asegurando que toda la lógica se ejecute solo después de que el DOM esté completamente cargado. Esto garantiza que los elementos necesarios estén disponibles antes de intentar manipularlos.

Búsqueda del botón:

- Una vez que el DOM está listo, el código busca un slot llamado "boton-enviar" dentro del shadow root del componente. Este slot permite que un botón definido en el light DOM (el DOM principal) sea proyectado al shadow DOM del componente. Si hay un elemento asignado al slot, lo usa; de lo contrario, busca un botón dentro del slot en el shadow DOM.

Listener del botón:

- Si se encuentra un botón (ya sea del light DOM o del shadow DOM), se le añade un listener para el evento click. Esto significa que la lógica principal se disparará cuando el usuario haga clic en el botón.

Validación de entrada:

- Al hacer clic en el botón, el código obtiene el valor de un campo de entrada (<input>) con el id "nombreNuevo" dentro del shadow root, eliminando espacios en blanco con trim(). Luego, realiza dos validaciones:
- Si el valor está vacío, muestra un mensaje de error en la consola y una alerta al usuario, deteniendo la ejecución.
- Si el valor tiene menos de 3 caracteres, también muestra un error en la consola y una alerta, deteniendo la ejecución.

Actualización de otro componente:

- Si el valor pasa las validaciones, el código busca un elemento personalizado en el documento llamado <tarjeta-usuario>. Si lo encuentra, busca un slot llamado "nombre" dentro del shadow root de ese elemento. Luego, intenta obtener el primer elemento asignado a ese slot (es decir, un elemento del light DOM proyectado en él). Si este elemento existe, actualiza su contenido de texto con el valor ingresado por el usuario y registra un mensaje de éxito en la consola.

Manejo de errores:

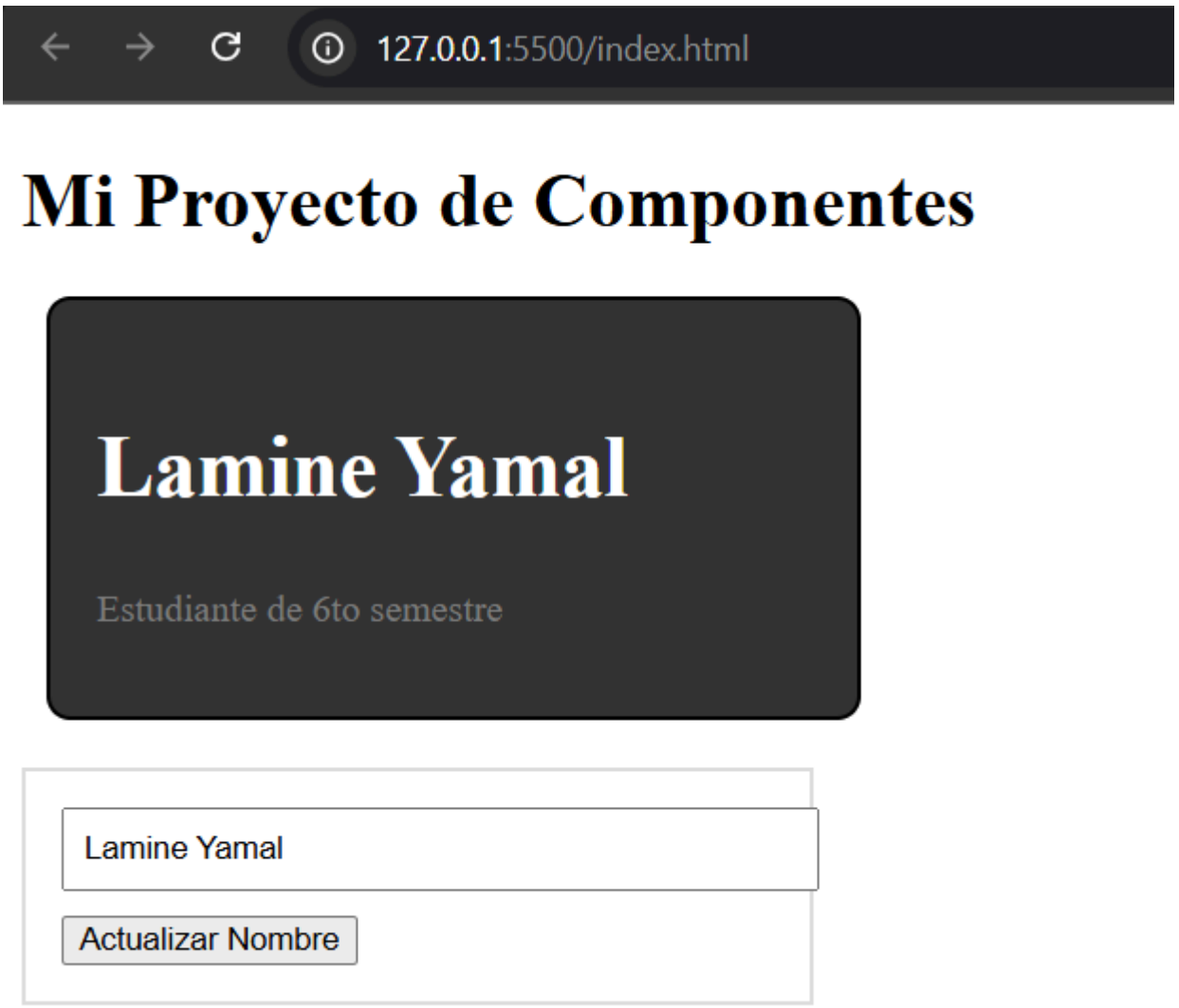
- Si alguna parte del proceso falla —como no encontrar el botón, el componente <tarjeta-usuario>, el slot "nombre", o el elemento asignado al slot— se muestra un mensaje de error descriptivo en la consola.

Ejemplo

La imagen muestra el componente <formulario-actualizar> actualizando el nombre que ya estaba asignado.

Figura 5

Actualización del componente <formulario-actualizar> con un nuevo nombre



Nota: Elaboración propia (2025). La imagen presenta el componente funcionando en un navegador.

4. CONCLUSIONES

La realización de la Tarea 2 en el laboratorio de Programación Integrativa permitió cumplir con los objetivos establecidos, fortaleciendo el aprendizaje en el uso de estándares web modernos. En primer lugar, se logró desarrollar un componente web, <formulario-actualizar>, que integra elementos HTML nativos y slots, permitiendo la personalización del contenido y la interacción con el componente <tarjeta-usuario> de la Tarea 1.

La implementación de validaciones básicas en el formulario, como verificar que el nombre ingresado no esté vacío y tenga al menos tres caracteres, aseguró una funcionalidad más robusta y alineada con los estándares de usabilidad. Asimismo, se estableció comunicación entre los componentes mediante document.querySelector, aunque enfrentamos desafíos de sincronización del DOM que fueron resueltos con el evento DOMContentLoaded, garantizando que las modificaciones dinámicas se realizarán correctamente. Por otra parte, la gestión del proyecto en GitHub, mediante la creación de la rama tarea2-integracion-html y la documentación con capturas en la carpeta docs/, reflejó un manejo disciplinado y organizado.