UNIVERSIDAD DE LAS FUERZAS ARMADAS "ESPE"
CIENCIAS DE LA COMPUTACIÓN
PROGRAMACION INTEGRATIVA DE COMPONENTES WEB

Nombre: Andrés Revelo
NRC: 16496
Fecha: 20/ Junio/2024

Aplicaciones para Web Components

- **Introducción**

En este manual, exploraremos el desarrollo de una aplicación web utilizando Web Components, un estándar del W3C que permite crear elementos personalizados y reutilizables en la web. Los Web Components están diseñados para facilitar la reutilización de código, el encapsulamiento y la interoperabilidad en aplicaciones web modernas.

- **Estructura del Proyecto**

La estructura de directorios y archivos para el proyecto es la siguiente:

```
tarea3/
├── index.html
├── css/
│   └── styles.css
├── js/
│   ├── app.js
│   ├── task.js
│   └── task-list.js
```

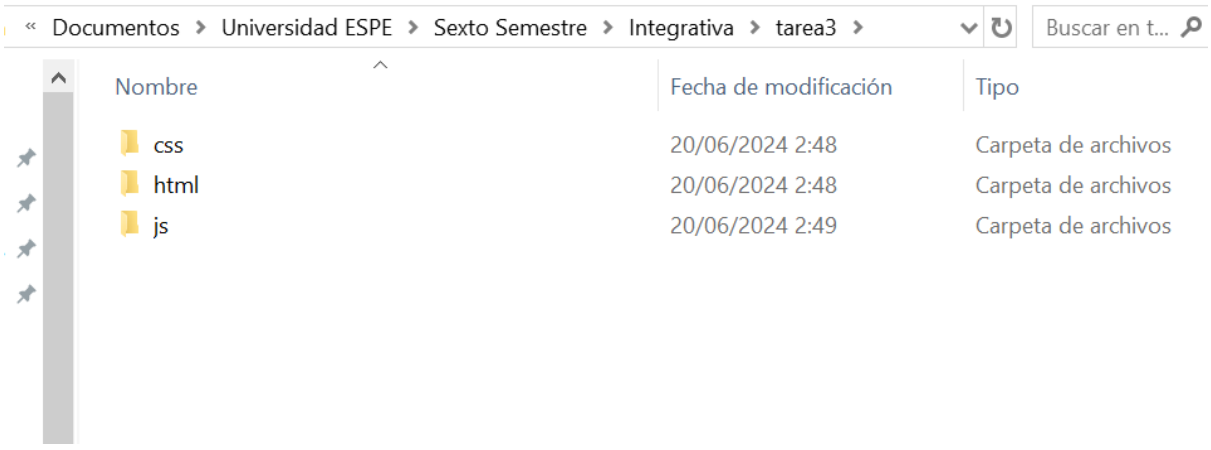

Ilustración 1 Directorio de carpetas

- Desarrollo

Para esta tarea, desarrollaremos una aplicación web sencilla que permitirá gestionar una lista de tareas (to-do list) utilizando Web Components. La aplicación demostrará el uso de componentes reutilizables, el ciclo de vida de los componentes web, y principios de POO en JavaScript. La aplicación incluirá las funcionalidades básicas de agregar, marcar como completada y eliminar tareas.

En las siguientes secciones, detallaremos cada paso del desarrollo de la aplicación, desde la estructura del proyecto hasta la implementación de las funcionalidades utilizando HTML, CSS y JavaScript. Al seguir este manual, podrás adquirir una comprensión práctica de cómo aplicar los principios de OOP en el desarrollo de aplicaciones web con JavaScript.

-Código HTML

Desarrollamos nuestro código de HTML, esto para generar el aspecto visual de nuestra página, aquí definimos el contenido de igual forma. Este archivo define la estructura básica de la página, incluye el título, el encabezado, un botón interactivo y un pie de página. También enlaza los archivos CSS y JavaScript.

html > index.html

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>To-Do App</title>
    <link rel="stylesheet" href="../css/styles.css">
</head>
<body>
    <div class="container">
        <header>
            <h1>Lista de Tareas</h1>
        </header>
        <main>
            <task-list></task-list>
        </main>
        <footer>
            <p>&copy; 2024 To-Do App</p>
        </footer>
    </div>
    <script src="../js/task.js" type="module"></script>
    <script src="../js/task-list.js" type="module"></script>
    <script src="..//js/app.js" type="module"></script>
</body>
</html>
```

Ilustración 2 Código HTML

-Código CSS

Este archivo define estilos básicos para el cuerpo, encabezado, contenido principal y pie de página. Se utiliza CSS simple para mantener el diseño claro y accesible.

Bienvenido | # styles.css | index.html | app.js

css > # styles.css > ...

```css
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    background-color: #f2f2f2;
}

.container {
    width: 80%;
    margin: 20px auto;
    padding: 20px;
    background-color: #fff;
    box-shadow: 0 0 10px rgba(0,0,0,0.1);
    border-radius: 8px;
}

header {
    background-color: #007bff;
    color: #fff;
    padding: 15px 20px;
    text-align: center;
    border-radius: 8px 8px 0 0;
    margin-bottom: 20px;
}

main {
    background-color: #f8f9fa;
    padding: 20px;
    border-radius: 8px;
```

Ilustración 3 Código CSS

-Interacción con JavaScript

Archivo task.js

Definimos nuestra clase taskItem, que extiende de HTMLElement, lo que significa que estamos creando un nuevo tipo de elemento HTML.

También creamos nuestro constructor, el shadowDOM, render actualiza el contenido

Ilustración 4 Código JavaScript

attributeChangedCallback(name, oldValue, newValue):

Este método se llama cuando uno de los atributos observados (especificados en observedAttributes) cambia.

name: Nombre del atributo que cambió.

oldValue: El valor antiguo del atributo.

newValue: El nuevo valor del atributo.

if (name === 'completed' && this.shadowRoot):

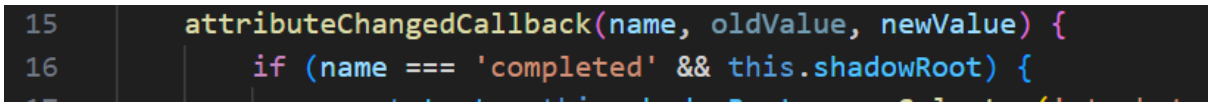

Ilustración 5

Comprobamos si el nombre del atributo que cambió es completed y si el shadow DOM está presente (this.shadowRoot).

Selecciona el elemento con la clase task-text dentro del shadow DOM.

Si el nuevo valor del atributo completed es 'true', añadimos la clase task-completed al elemento text.

Si el nuevo valor no es 'true', removemos la clase task-completed del elemento text.

La clase task-completed probablemente aplica estilos que indican que la tarea está completada, como el tachado del texto.

```
17              const text = this.shadowRoot.querySelector('.task-text');
18              if (newValue === 'true') {
19                  text.classList.add('task-completed');
20              } else {
21                  text.classList.remove('task-completed');
22              }
23          }
24      }
```

Ilustración 6

El método render es responsable de renderizar o actualizar el contenido del shadow DOM, TaskItem. Este atributo contiene el texto de la tarea, obtiene el valor del atributo completed y lo compara con la cadena ‘true’. Si el valor del atributo es ‘true’, la variable completed será true (un booleano). De lo contrario, completed será false.

```
26      render() {
27          const taskText = this.getAttribute('text');
28          const completed = this.getAttribute('completed') === 'true';
29
30          this.shadowRoot.innerHTML = `
```

Ilustración 7

Archivo task-list.js

Aquí definimos una clase TaskList que extiende de HTMLElement. Al crear una instancia de TaskList, se establece un "shadow DOM" para encapsular el HTML y el CSS del componente, evitando conflictos con otros estilos del documento. La lista de tareas se inicializa como un arreglo vacío. Cuando el componente se añade al DOM, se llama al método connectedCallback, que a su vez llama

a render para dibujar el componente en la página. En el método render, se define la estructura HTML y los estilos del componente dentro del shadow DOM.

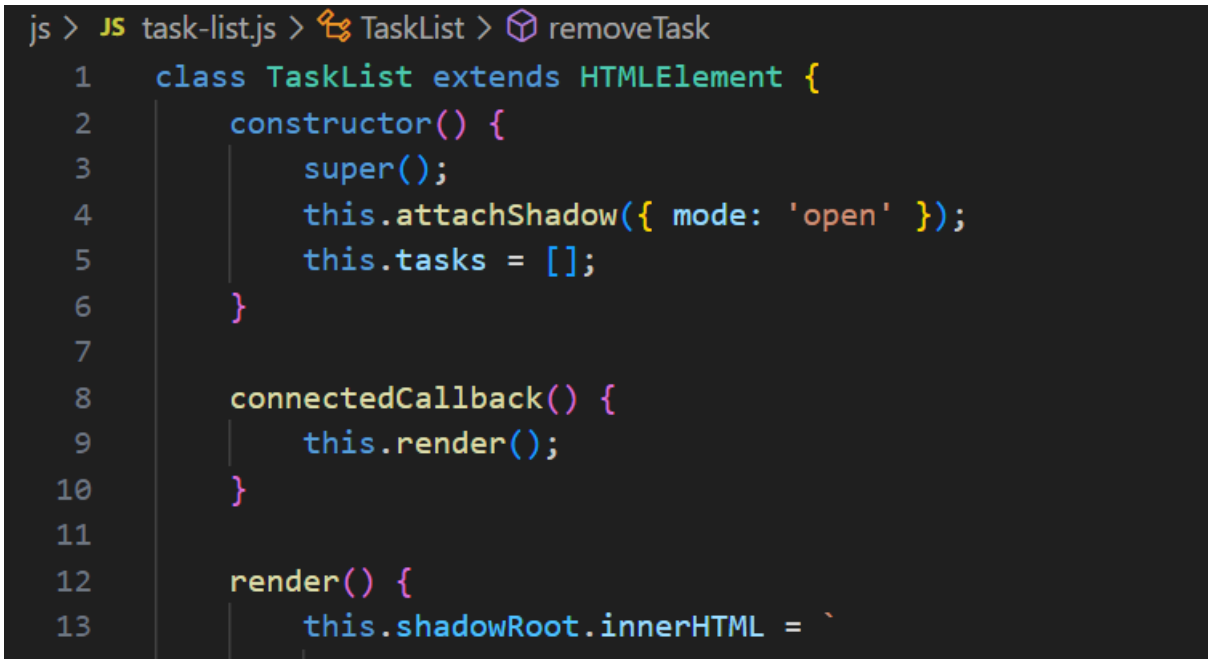

Ilustración 8

En esta parte del código, se agregan event listeners a los botones dentro del shadow DOM del componente. Para cada botón de completar (.complete), se añade un listener que llama a toggleComplete con el índice de la tarea correspondiente. De manera similar, para cada botón de eliminar (.delete), se añade un listener que llama a removeTask con el índice de la tarea. Además, se añade un listener al botón de "Agregar Tarea" (#addTaskBtn) que llama a addTask cuando se hace clic en él. Estos listeners son cruciales para manejar las interacciones del usuario con el componente.

Ilustración 9

Estos métodos gestionan el estado de las tareas. toggleComplete invierte el estado de completado de la tarea en el índice dado y vuelve a renderizar el componente para reflejar el cambio. removeTask elimina la tarea en el índice dado del arreglo tasks y vuelve a renderizar el componente. Al volver a llamar a render, aseguramos que la interfaz se actualice para mostrar el nuevo estado de las tareas.

```
107     toggleComplete(index) {
108         const task = this.tasks[index];
109         task.completed = !task.completed;
110         this.render();
111     }
112
113     removeTask(index) {
114         this.tasks.splice(index, 1);
115         this.render();
116     }
```

Ilustración 10

El método addTask se encarga de añadir una nueva tarea a la lista. Obtiene el valor del input de nueva tarea (#newTask), y si el texto no está vacío, crea una nueva tarea con el texto proporcionado y un estado de completado false, añadiéndola al arreglo tasks. Luego, limpia el input y vuelve a renderizar el componente para mostrar la nueva tarea en la lista. Finalmente, customElements.define('task-list', TaskList) registra el nuevo elemento task-list para que pueda ser usado en el HTML de la página.

```
118     addTask() {
119         const input = this.shadowRoot.querySelector('#newTask');
120         const taskText = input.value.trim();
121
122         if (taskText) {
123             this.tasks.push({ text: taskText, completed: false });
124             input.value = '';
125             this.render();
126         }
127     }
128 }
129
130 customElements.define('task-list', TaskList);
131
```

Ilustración 11

-Ejecución de la Página Web

Agregamos diferentes tareas a realizar

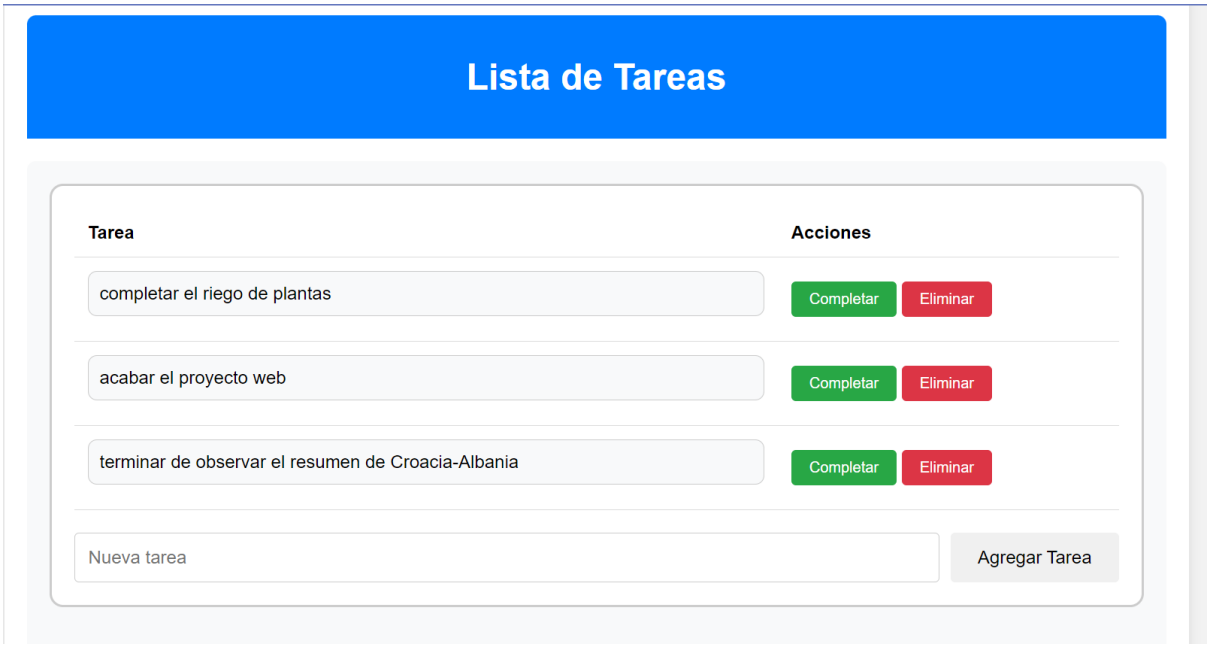

Ilustración 12 Ejecución de Programa

Activamos la opción completar que se nos marca la acción que hemos elegido

Ilustración 13 Acción Completar

Ahora activamos la opción eliminar lo cual nos permite eliminar una actividad que ya no deseemos

Ilustración 14 Función Eliminar

- **Conclusión**

-Los Web Components permiten encapsular la lógica y el estilo en componentes reutilizables, mejorando la modularidad y mantenibilidad del código.

-Al seguir los estándares del W3C, los Web Components garantizan la interoperabilidad y la posibilidad de integrar componentes personalizados en cualquier proyecto web sin conflictos de estilo o comportamiento.