UNIVERSIDAD DE LAS FUERZAS ARMADAS – ESPE

Departamento de Ciencias de la Computación

Carrera de Ingeniería en Tecnologías de la Información

Proyecto Final

"AirGuard - Plataforma Educativa de Monitoreo de Calidad del Aire"

Asignatura: Programación Integrativa de Componentes

NRC: 21602

Docente: Ing. ANGEL GEOVANNY CUDCO POMAGUALLI

Autores:

• Josue Zambrano

• Cesar Arico

• Alan Herrera

Sangolquí – Ecuador

27 de May de 2025

Contenido

1. Descripción del Problema

La contaminación atmosférica es una amenaza creciente en zonas urbanas. Según la OMS, más del 90% de la población mundial respira aire con altos niveles de contaminantes como partículas PM2.5 y PM10, que pueden provocar enfermedades respiratorias, cardiovasculares y muertes prematuras. A pesar de esto, la mayoría de personas no tiene acceso a información clara y en tiempo real sobre la calidad del aire en su entorno.

2. Objetivo General

Desarrollar una SPA (Single Page Application) moderna que eduque a los usuarios sobre la contaminación atmosférica, visualice datos simulados de calidad del aire en tiempo real mediante un dashboard interactivo y brinde recomendaciones personalizadas para reducir la exposición a contaminantes, usando Web Components, Shadow DOM y arquitectura modular.

3. Requisitos Técnicos y Funcionales

• Implementación de al menos 5 Custom Elements con Shadow DOM (encapsulamiento total de lógica y estilos).

• Uso de métodos de ciclo de vida como connectedCallback para inicialización de eventos y datos.

• Consumo de APIs externas (OpenWeatherMap o JSON-server simulado) para obtener niveles de PM2.5 y PM10, visualizados con Chart.js.

• CRUD completo basado en arrays, gestionado desde el componente <data-crud>.

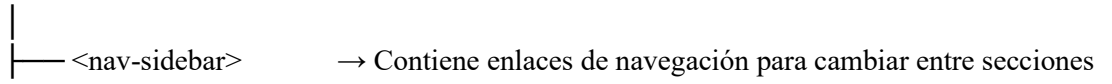
• Interfaz dinámica con animaciones CSS y diseño UI/UX moderno (sidebar, menús, sección educativa).

• Métodos render() para actualización dinámica y uso de arrow functions para eventos (ej. formularios, filtros).

• Sección educativa con artículos e infografías generadas desde arrays de objetos.

4. Diagrama de Componentes y Explicación Técnica

El proyecto se estructura mediante Web Components personalizados, cada uno con su propia lógica encapsulada gracias al uso de Shadow DOM. Esto permite que cada componente sea autónomo, fácilmente mantenible y reutilizable.

index_unificado.html


│
├── <main-menu> → Barra superior que gestiona login/logout y navegación principal
│
├── <air-quality-dashboard> → Componente principal que gestiona los datos de calidad del aire
│   └── <air-quality-chart> → Visualiza los datos usando Chart.js (PM2.5, PM10, NO2, etc.)
│
├── <data-crud> → Gestiona las recomendaciones usando arrays (añadir, editar, eliminar)
│
└── <user-recommendations> → Presenta sugerencias personalizadas para protección contra contaminantes

Cada componente define su estructura interna y estilos usando el método attachShadow({mode: 'open'}). Esto garantiza aislamiento completo del CSS y reduce los conflictos con otros estilos globales. Los métodos render() y connectedCallback() son claves para actualizar el contenido dinámicamente cuando cambia el estado interno del componente.

5. Ejemplo Técnico del Sistema

A continuación, se presenta un ejemplo del componente <air-quality-dashboard>, el cual representa el panel principal de monitoreo de calidad del aire. Este componente utiliza Shadow DOM para encapsular su lógica y estilos, consume una API simulada, y aplica renderizado dinámico mediante métodos asincrónicos y funciones modernas.

Fragmento del componente AirQualityDashboard.js:

```
import { AirQualityService } from '../api/airQualityService.js';

// Definición de clase personalizada
class AirQualityDashboard extends HTMLElement {
  // Constructor para inicializar propiedades y estado
    constructor() {
        super();
        this.attachShadow({ mode: 'open' });
// Insertamos estilos modernos al componente
const style = document.createElement('style');
        this.city = 'London';
        this.country = 'uk';
    }
  // Método invocado cuando el componente se inserta en el DOM
    connectedCallback() {
        this.renderDashboard();
    }
    async renderDashboard() {
        const weather = await
AirQualityService.fetchCurrentWeather(this.city, this.country);
        const airQuality = await
AirQualityService.fetchAirQualityData(
```

```
            weather.coord.lat,
            weather.coord.lon
        );
        this.shadowRoot.innerHTML =
            <div class="city-selector">
                <label for="citySelect">Ciudad:</label>
                <select id="citySelect">
                    <option value="London,uk"${this.city ===
'London' ? ' selected' : ''}>Londres</option>
                    <option value="Quito,ec"${this.city ===
'Quito' ? ' selected' : ''}>Quito</option>
                    <option value="New York,us"${this.city ===
'New York' ? ' selected' : ''}>New York</option>
                    <option value="Madrid,es"${this.city ===
'Madrid' ? ' selected' : ''}>Madrid</option>
                    <option value="Tokyo,jp"${this.city ===
'Tokyo' ? ' selected' : ''}>Tokyo</option>
                </select>
            </div>
            <div class="dashboard">
                <div class="section">
                    <h2>Clima actual en ${weather.name}</h2>
                    <div><span class="label">Temperatura:</span>
${weather.main.temp}°C</div>
                    <div><span class="label">Condición:</span>
${weather.weather[0].description}</div>
                    <div><span class="label">Humedad:</span>
${weather.main.humidity}%</div>
                </div>
                <div class="section">
                    <h2>Calidad del Aire</h2>
                    <div class="aqi ${airQuality.aqi.level}">
                        AQI: ${airQuality.aqi.value}
(${this.getAQIText(airQuality.aqi.level)})
                    </div>
                    <div class="pollutants">
                        <div class="pollutant"><span
class="label">PM2.5:</span> ${airQuality.pollutants.pm25}
µg/m³</div>
                        <div class="pollutant"><span
class="label">PM10:</span> ${airQuality.pollutants.pm10}
µg/m³</div>
                        <div class="pollutant"><span
class="label">CO:</span> ${airQuality.pollutants.co} µg/m³</div>
                        <div class="pollutant"><span
class="label">NO₂:</span> ${airQuality.pollutants.no2}
µg/m³</div>
                        <div class="pollutant"><span
class="label">O₃:</sp
```

Consumo de APIs Simuladas

El componente utiliza un servicio llamado AirQualityService que encapsula la lógica para obtener datos de clima y calidad del aire. Se utiliza programación asincrónica con async/await para

realizar solicitudes simuladas a una API como OpenWeatherMap o un servidor JSON local con JSON-server.

Fragmento del archivo airQualityService.js:

```
const API_KEY = '9ef902745a434a82e43a393bc1751a4e';
export class AirQualityService {
    static async fetchCurrentWeather(city = 'London', country =
'uk') {
        try {
  // Solicitud fetch para obtener datos del servidor
            const response = await fetch(

`https://api.openweathermap.org/data/2.5/weather?q=${city},${coun
try}&appid=${API_KEY}&units=metric`
            );

            if (!response.ok) {
                throw new Error(`HTTP error! status:
${response.status}`);
            }

            return await response.json();
        } catch (error) {
            console.error('Error fetching current weather:',
error);
            return this.getMockWeatherData();
        }
    }

    static async fetchAirQualityData(lat = 51.5085, lon = -
0.1257) {
        try {
  // Solicitud fetch para obtener datos del servidor
            const response = await fetch(

`https://api.openweathermap.org/data/2.5/air_pollution?lat=${lat}
&lon=${lon}&appid=${API_KEY}`
            );
            if (!response.ok) {
                console.log('API Key no activa o error, usando
datos de prueba');
                return this.getMockAirQualityData();
            }

            const data = await response.json();
            return this.processAirQualityData(data);
```

```
        } catch (error) {
            console.error('Error fetching air quality data:',
error);
            return this.getMockAirQualityData();
        }
    }

    static processAirQualityData(data) {
        const mainPollutant = data.list[0].main.aqi;
        const components = data.list[0].components;

        return {
            aqi: {
                value: mainPollutant,
                level: this.getAQILevel(mainPollutant)
            },
            pollutants: {
                pm25: components.pm2_5,
                pm10: components.pm10,
                co: components.co,
                no2: components.no2,
                o3: components.o3,
                so2: components.so2
            },
            lastUpdated: new Date().toISOString()
        };
    }

    /*static getMockWeatherData() {
        return {
            coord: { lon: -0.1257, lat: 51.5085 },
            weather: [{ id: 501, main: 'Rain', description:
'moderate rain', icon: '10d' }],
            main: {
                temp: 12.86,
                feels_like: 12.28,
                temp_min: 11.77,
                temp_max: 13.58,
                pressure: 1017,
                humidity: 80
            },
            visibility: 8000,
            wind: { speed: 3.6, deg: 310 },
            rain: { '1h': 1.68 },
            clouds: { all: 75 },
```

```
            dt: Math.floor(Date.now() / 1000),
            sys: {
                type: 2,
                id: 268730,
                country: 'GB',
                sunrise: Math.floor(Date.now() / 1000)
```

<nav-sidebar>

Este componente gestiona la navegación lateral de la aplicación, permitiendo al usuario acceder rápidamente a las secciones del dashboard, recomendaciones y la parte educativa.

```
// Definición de clase personalizada
class NavSidebar extends HTMLElement {
  // Constructor para inicializar propiedades y estado
    constructor() {
        super();
        this.attachShadow({ mode: 'open' });
// Insertamos estilos modernos al componente
const style = document.createElement('style');
    }
  // Renderiza el contenido del componente
    render() {
        this.shadowRoot.innerHTML =
        <nav class="sidebar">
                <div class="logo">AirGuard</div>
                <ul class="nav-list">
                    <li class="nav-item">
                        <a href="#dashboard">Dashboard</a>
                    </li>
                    <li class="nav-item active">
                        <a
href="#recomendaciones">Recomendaciones</a>
                    </li>
                    <li class="nav-item">
                        <a href="#educativo">Educativo</a>
                    </li>
                    <li class="nav-item">
                        <a href="#acerca">Acerca de</a>
                    </li>
                </ul>
            </nav>
        `;
    }
  // Método invocado cuando el componente se inserta en el DOM
    connectedCallback() {
  // Renderiza el contenido del componente
        this.render();
```

```
    }
}
customElements.define('nav-sidebar', NavSidebar);
```

<data-crud>

Permite al usuario agregar, editar o eliminar recomendaciones de protección personal. Usa arrays como base de datos y maneja los eventos de formulario con funciones flecha.

```
// Definición de clase personalizada
class DataCrud extends HTMLElement {
  // Constructor para inicializar propiedades y estado
    constructor() {
        super();
        this.attachShadow({ mode: 'open' });
        this.editingId = null; // Añadimos variable para
controlar el modo edición


// Insertamos estilos modernos al componente
const style = document.createElement('style');
style.textContent = `
  :host {
    display: block;
    font-family: 'Inter', sans-serif;
    background: #ffffff;
    color: #333;
    padding: 1rem;
    border-radius: 12px;
    box-shadow: 0 2px 10px rgba(0,0,0,0.05);
    margin-bottom: 1rem;
  }
  h2, h3 {
    color: #1E2A38;
  }
  p {
    color: #555;
  }
`;
this.shadowRoot.appendChild(style);

        // Cargar datos desde localStorage o usar datos por
defecto
        this.recommendations =
JSON.parse(localStorage.getItem('recommendations')) || [
            { id: 1, title: 'Usar mascarilla', description: 'En
días de alta contaminación' },
            { id: 2, title: 'Evitar ejercicio al aire libre',
```

```
description: 'Cuando el índice de calidad del aire es malo' }
        ];
    }

  // Método invocado cuando el componente se inserta en el DOM
    connectedCallback() {
  // Renderiza el contenido del componente
        this.render();
  // Asignación de eventos a elementos DOM
        this.addEventListeners(); // Cambiamos
setupEventListeners por addEventListeners
    }

  // Renderiza el contenido del componente
    render() {
        this.shadowRoot.innerHTML = `
            <style>
                .crud-container {
                    padding: 30px;
                    max-width: 800px;
                    margin: 0 auto;
                    background: #ffffff;
                    border-radius: 12px;
                    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
                }
                .recommendation-form {
                    display: flex;
```

<data-crud>

Este componente permite realizar operaciones CRUD sobre recomendaciones personalizadas de protección. Utiliza arrays como estructura de almacenamiento y localStorage para persistencia.

```
// Definición de clase personalizada
class DataCrud extends HTMLElement {
  // Constructor para inicializar propiedades y estado
    constructor() {
        super();
        this.attachShadow({ mode: 'open' });
        this.editingId = null; // Añadimos variable para
controlar el modo edición


// Insertamos estilos modernos al componente
const style = document.createElement('style');
```

```
        // Cargar datos desde localStorage o usar datos por
defecto
        this.recommendations =
JSON.parse(localStorage.getItem('recommendations')) || [
            { id: 1, title: 'Usar mascarilla', description: 'En
días de alta contaminación' },
            { id: 2, title: 'Evitar ejercicio al aire libre',
description: 'Cuando el índice de calidad del aire es malo' }
        ];
    }

  // Método invocado cuando el componente se inserta en el DOM
    connectedCallback() {
  // Renderiza el contenido del componente
        this.render();
  // Asignación de eventos a elementos DOM
        this.addEventListeners(); // Cambiamos
setupEventListeners por addEventListeners
    }

  // Renderiza el contenido del componente
    render() {
        this.shadowRoot.innerHTML = `

            <div class="crud-container">
                <h2>Recomendaciones de Protección</h2>
                <form class="recommendation-form">
                    <input type="text" id="title"
placeholder="Título" required>
                    <textarea id="description"
placeholder="Descripción" required></textarea>
                    <button type="submit">${this.editingId ?
'Actualizar' : 'Agregar'} Recomendación</button>
                </form>
                <div class="recommendations-list">
                    ${this.recommendations.map(rec => `
                        <div class="recommendation-card">
                            <h3>${rec.title}</h3>
                            <p>${rec.description}</p>
                            <div class="actions">
                                <button class="edit-btn" data-
id="${rec.id}">Editar</button>
                                <button class="delete-btn" data-
id="${rec.id}">Eliminar</button>
                            </div>
```

```
                        </div>
                    `).join('')}
                </div>
            </div>
        `;
    }

  // Asignación de eventos a elementos DOM
    addEventListeners() {
        const form =
this.shadowRoot.querySelector('.recommendation-form');
        form.addEventListener('submit', (e) => {
            e.preventDefault(); // Previene el comportamiento por
defecto

            const title =
this.shadowRoot.querySelector('#title').value.trim();
            const description =
this.shadowRoot.querySelector('#description').value.trim();

            if (!title || !description) return; // Validación
básica

            if (this.editingId) {
                this.updateRecommendat
```

<user-recommendations>

Este componente es responsable de mostrar las recomendaciones del usuario en una interfaz visual organizada. Integra internamente el componente <data-crud> para gestionar los datos.

```
// Definición de clase personalizada
class UserRecommendations extends HTMLElement {
  // Constructor para inicializar propiedades y estado
    constructor() {
        super();
        this.attachShadow({ mode: 'open' });
// Insertamos estilos modernos al componente
const style = document.createElement('style');
    }
  // Método invocado cuando el componente se inserta en el DOM
    connectedCallback() {
  // Renderiza el contenido del componente
        this.render();
    }
  // Renderiza el contenido del componente
    render() {
        this.shadowRoot.innerHTML = `
            <div class="recommendations-container">
```

```
                <h2>Recomendaciones Personalizadas</h2>
                <data-crud></data-crud>
            </div>
        `;
    }
}
customElements.define('user-recommendations',
UserRecommendations);
```

<educate-section>

El componente <educate-section> genera dinámicamente contenido educativo sobre los contaminantes PM2.5 y PM10. Usa arrays de objetos y métodos de renderizado para desplegar tarjetas informativas e imágenes asociadas.

```
// Definición de clase personalizada

class EducateSection extends HTMLElement {

  // Constructor para inicializar propiedades y estado

  constructor() {

    super();

    // Crea un Shadow DOM para encapsular los estilos y
estructura

    this.attachShadow({ mode: 'open' });

// Insertamos estilos modernos al componente

const style = document.createElement('style');









    this.initializeData();

  }

// Inicializa los datos del componente con información sobre
PM2.5

  initializeData() {

    // Array de objetos con información principal sobre PM2.5

    this.content = [

      {

      titulo: "¿Qué es el PM2.5 y PM10?",
```

texto: "El PM2.5 y PM10 son partículas microscópicas suspendidas en el aire. El PM2.5 tiene un diámetro menor a 2.5 micras y el PM10 menor a 10 micras. Ambas pueden penetrar en el sistema respiratorio, pero el PM2.5 llega más profundo y es más dañino para la salud. Estas partículas pueden estar compuestas de polvo, hollín, metales pesados y compuestos orgánicos.",

img: "img/1.webp",

alt: "Contaminación ambiental"

},

{

titulo: "Efectos en la salud",

texto: "La exposición prolongada a PM2.5 y PM10 puede causar problemas respiratorios, cardiovasculares y aumentar la mortalidad. El PM2.5 es especialmente peligroso por su capacidad de ingresar al torrente sanguíneo. Estudios han demostrado que los niños y adultos mayores son los más vulnerables, y la contaminación puede agravar enfermedades como el asma y la bronquitis.",

img: "img/2.png",

alt: "Salud pulmonar"

},

{

titulo: "¿Cómo protegerse?",

texto: "Usar mascarillas, evitar actividades al aire libre en días contaminados y mantener buena ventilación ayuda a reducir la exposición tanto a PM2.5 como a PM10. Además, el uso de purificadores de aire con filtros HEPA en interiores puede disminuir significativamente la concentración de partículas nocivas.",

img: "img/3.jpg",

alt: "Persona usando mascarilla"

},

{

titulo: "¿Dónde se mide el PM2.5 y PM10?",

texto: "Se mide en estaciones de monitoreo ambiental y con sensores personales. Consulta los índices de calidad del aire en tu ciudad para ambos tipos de partículas. Algunas ciudades publican estos datos en tiempo real a través de aplicaciones móviles y sitios web oficiales.",

```
      img: "img/4.jpg",

      alt: "Estación de monitoreo ambiental"

      },

      {

      titulo: "Monitoreo en Tiempo Real",

      texto: "Las tecnologías modernas permiten monitorear la
calidad del aire en tiempo real, tanto para PM2.5 como para PM10.
Utiliza apps y dispositivos inteligentes para mantenerte
informado. Algunos sensores pueden integrarse en sistemas de
domótica para activar alertas o purificadores automáticamente.",

      img: "img/5.jpg",

      alt: "Tecnología de monitoreo"

      },

      {
```

6. Conclusiones

- La utilización de Web Components permitió desarrollar una aplicación modular, donde cada funcionalidad está encapsulada en componentes reutilizables. Esto mejora la mantenibilidad del código y facilita futuras ampliaciones.
- La integración de Chart.js y APIs externas simuladas posibilitó representar de forma clara e interactiva los datos de calidad del aire (PM2.5 y PM10), mejorando la comprensión del usuario sobre el impacto de la contaminación.
- La implementación de operaciones CRUD sobre recomendaciones, junto con una sección educativa, permitió al usuario interactuar activamente con la plataforma y recibir sugerencias personalizadas, reforzando el componente pedagógico del sistema.
- El uso de ES6+, funciones flecha, manejo de eventos y programación asincrónica fomentó la adopción de buenas prácticas en el desarrollo frontend, alineadas con estándares modernos de JavaScript.

7. Recomendaciones

- Integrar una base de datos real o backend con Node.js permitiría persistir los datos de recomendaciones más allá del almacenamiento local, facilitando la escalabilidad a múltiples usuarios.
- En futuras versiones, se recomienda utilizar servicios como IQAir o OpenAQ para obtener datos en tiempo real de calidad del aire a nivel global, aumentando la precisión de la información mostrada.
- Se puede optimizar la carga de componentes mediante lazy loading y mejorar la accesibilidad con etiquetas ARIA y navegación por teclado.
- Para entornos de producción, sería útil implementar pruebas automatizadas (unitarias e integración) y desplegar la aplicación en plataformas como Vercel o Netlify.

8. Bibliografía

- Organización Mundial de la Salud. (2021). La contaminación del aire. Recuperado de: https://www.who.int/es/news-room/fact-sheets/detail/ambient-(outdoor)-air-quality-and-health
- Mozilla Developer Network. (2023). Web Components - MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/Web_Components
- Chart.js. (2023). Documentation. https://www.chartjs.org/docs/latest/
- Google Developers. (2022). JavaScript ES6+ Features. https://developers.google.com/web/updates/2015/03/es6-modules-today
- OpenAQ. (2024). Open Air Quality Platform. https://openaq.org
- IQAir. (2024). World Air Quality Index. https://www.iqair.com/world-air-quality