Evidencias de Implementación de JavaScript y APIs

Proyecto Sazón de Finca

Alejandra Solórzano Medrano

Cédula: 3-0540-0768

Universidad Técnica Nacional

ISW-512 Diseño de Aplicaciones Web

Mit. Sergio González Salazar

Resumen

El presente documento detalla las evidencias de implementación de JavaScript, consumo de APIs y uso de bibliotecas externas en el proyecto web "Sazón de Finca". Se documenta la aplicación de código JavaScript propio en seis archivos diferentes, el consumo de tres archivos JSON como API REST, la integración de Google Maps API como servicio externo, y la implementación de jQuery para validaciones en tiempo real. El proyecto cumple con todos los requerimientos técnicos establecidos para el curso ISW-512 Diseño de Aplicaciones Web.

Palabras clave: JavaScript, API REST, jQuery, validaciones, carrito de compras, Google Maps

1. Validaciones e Interacciones con JavaScript Propio

1.1 Formulario de Contacto (sendEmail.js)

El archivo sendEmail.js implementa la función calcularEdad() que realiza validaciones exhaustivas del formulario de contacto antes del envío. Esta función valida que los campos de nombre y apellido no contengan caracteres numéricos, verifica el formato del correo electrónico mediante expresiones regulares, y calcula automáticamente la edad del usuario basándose en su fecha de nacimiento (González, 2024).

Las validaciones implementadas incluyen:

- Verificación de campos vacíos (nombre, apellido, correo, mensaje)
- Validación de ausencia de números en nombre y apellido mediante la expresión regular /\d/
- Validación de formato de correo electrónico con /^[^\s@]+@[^\s@]+\.[^\s@]+$/
- Validación de fecha de nacimiento considerando fechas futuras y rangos válidos (0-120 años)
- Cálculo de edad considerando año, mes y día actuales
- Almacenamiento de edad calculada en campo oculto del formulario

El archivo contiene 87 líneas de código JavaScript puro y se ejecuta al momento del envío del formulario mediante el evento onclick del botón de envío.

1.2 Sistema de Geolocalización (ubicacion.js)

El archivo ubicacion.js contiene la función initMap() que inicializa un mapa interactivo utilizando la API de Google Maps. La implementación establece las coordenadas geográficas precisas del negocio (latitud: 10.356647, longitud: -83.920565), configura el nivel de zoom en 14 para una visualización óptima, y coloca un marcador personalizado con el título "Sazon de Finca SA" (Google Developers, 2024).

La configuración incluye:

- Desactivación del control de pantalla completa (fullscreenControl: false)
- Activación del control de zoom (zoomControl: true)
- Desactivación de vista de calle (streetViewControl: false)
- Centrado automático del mapa en las coordenadas especificadas

El archivo consta de 19 líneas de código y se carga de forma asíncrona mediante el atributo defer en la etiqueta script del HTML.

1.3 Gestión de Productos (products.js)

El sistema de gestión de productos implementado en products.js utiliza programación asíncrona para cargar datos desde un archivo JSON. Las funciones principales incluyen cargarProductos() que utiliza la API Fetch para obtener datos, renderizarProductos() que genera dinámicamente elementos HTML, agregarProducto(id) que integra productos al carrito de compras, y verDetalle(id) que crea modales informativos de forma dinámica (MDN Web Docs, 2024).

Características técnicas:

- Uso de async/await para operaciones asíncronas

- Manipulación del DOM mediante createElement() y innerHTML
- Implementación de sistema de ordenamiento con tres criterios (alfabético, precio ascendente, precio descendente)
- Generación dinámica de modales que se crean y destruyen según necesidad
- Integración completa con el sistema de carrito de compras

El archivo contiene 98 líneas de código y utiliza event listeners para manejar la interacción del usuario con los controles de ordenamiento.

1.4 Galería Dinámica de Imágenes (gallery.js)

El archivo gallery.js implementa un sistema de galería completamente dinámico. La función cargarImagenes() obtiene datos mediante fetch asíncrono, mientras que mostrarImagenes() itera sobre el array de imágenes utilizando forEach() y crea dinámicamente los elementos HTML necesarios mediante document.createElement() (Flanagan, 2020).

Implementación técnica:

- Uso del selector document.querySelector(".row.g-4") para obtener el contenedor
- Creación dinámica de elementos div con clases responsive de Bootstrap
- Asignación de atributos src, alt, title y className a cada imagen
- Inserción en el DOM mediante appendChild()
- Sistema de clases responsive: col-12 col-sm-6 col-md-4 col-lg-3

El sistema consta de 26 líneas de código y carga 19 imágenes del proyecto de forma automática.

1.5 Sistema de Testimonios (testimonials.js)

El sistema de testimonios implementa funcionalidad de paginación manual y generación dinámica de elementos de calificación. Las funciones principales son cargarTestimonios() para obtención de datos, generarEstrellas(calificacion) que crea íconos visuales según rating, renderizarTestimonios() que muestra testimonios con paginación, verMasTestimonios() que carga seis testimonios adicionales, y ocultarTestimonios() que restablece la vista inicial (Haverbeke, 2018).

Características de implementación:

- Sistema de paginación implementado manualmente sin librerías externas
- Uso de Array.slice() para controlar cantidad de elementos mostrados
- Generación condicional de íconos de estrellas (bi-star-fill vs bi-star)
- Implementación de scroll suave mediante scrollIntoView({ behavior: 'smooth' })
- Control dinámico de visibilidad de botones según estado de paginación

El archivo consta de 92 líneas de código y maneja 12 testimonios de clientes con sistema de calificación de 1 a 5 estrellas.

1.6 Sistema de Carrito de Compras (cart.js)

El sistema de carrito representa la implementación JavaScript más completa del proyecto con 157 líneas de código. Utiliza sessionStorage para persistencia de datos durante la sesión del usuario e implementa once funciones principales que manejan todo el ciclo de vida del carrito de compras (Simpson, 2015).

Funciones implementadas:

- cargarCarrito(): Lee datos persistidos en sessionStorage y los convierte a objetos JavaScript
- guardarCarrito(): Serializa el array de productos y lo almacena en sessionStorage
- agregarAlCarrito(producto): Añade productos nuevos o incrementa cantidad de existentes
- eliminarDelCarrito(id): Filtra el array removiendo el producto especificado
- actualizarCantidad(id, cantidad): Modifica cantidades con validación de valores mínimos
- calcularTotal(): Utiliza reduce() para sumar todos los subtotales
- actualizarVistaCarrito(): Genera HTML dinámico para cada item del carrito
- enviarAWhatsApp(): Construye mensaje formateado y abre WhatsApp con URL codificada
- vaciarCarrito(): Limpia array y almacenamiento con confirmación del usuario

- mostrarNotificacion(mensaje): Crea elementos de notificación temporales
- toggleCarrito(): Controla visibilidad del modal mediante Bootstrap Modal API

Características técnicas avanzadas:

- Uso de operador spread (...producto) para clonación de objetos
- Implementación de find() para búsqueda eficiente en arrays
- Formateo de números con toLocaleString('es-CR') para separadores de miles
- Generación de templates literals complejos para HTML dinámico
- Integración con WhatsApp Business API mediante esquema wa.me
- Sistema de notificaciones con eliminación automática mediante setTimeout()

2. Consumo de API REST Externo

2.1 Google Maps JavaScript API

La integración de Google Maps API permite mostrar la ubicación geográfica exacta del negocio mediante un mapa interactivo. La implementación utiliza la versión semanal de la API cargada de forma asíncrona con callback a la función initMap() (Google Cloud, 2024).

Detalles de implementación:

URL de carga:

https://maps.googleapis.com/maps/api/js?key=AIzaSyAUxcBxVwUAXMHymqKvEJO_RH-9dlSj0gI&callback=initMap&libraries=&v=weekly

Configuración del mapa:

- Coordenadas: 10.356647°N, 83.920565°W
- Nivel de zoom: 14
- Controles personalizados según necesidades del usuario

Ubicación en el proyecto:

- Archivo HTML: index.html
- Sección: #ubication
- Archivo JavaScript: js/ubicacion.js
- Función: initMap()

La API proporciona funcionalidad de mapa interactivo actualizado automáticamente, interfaz familiar para usuarios, y permite navegación intuitiva mediante controles de zoom y desplazamiento.

3. Consumo de JSON como API REST

3.1 Sistema de Productos (products.json)

El archivo products.json contiene información estructurada de doce productos artesanales. Cada producto incluye identificador único, nombre comercial, presentación, precio en colones costarricenses, ruta de imagen, descripción detallada, y tipo de producto cuando aplica (Crockford, 2008).

Estructura de datos:

```
{
  "id": number,
  "nombre": string,
  "presentacion": string,
  "precio": number,
  "imagen": string,
  "descripcion": string,
  "tipo": string (opcional)
}
```

Proceso de consumo:

1. Función fetch() realiza petición HTTP GET al archivo JSON
2. Método .json() parsea la respuesta a objeto JavaScript

3. Datos se almacenan en variable global productos

4. Función renderizarProductos() itera sobre array y genera HTML

Aplicaciones en el proyecto:

- Catálogo completo de productos
- Sistema de ordenamiento múltiple
- Generación automática de tarjetas de producto
- Integración con carrito de compras
- Modal de detalles de producto

3.2 Sistema de Testimonios (testimonials.json)

El archivo testimonials.json almacena opiniones de clientes con estructura que incluye identificador, nombre del cliente, texto de opinión, producto específico comentado, y calificación numérica de 1 a 5 estrellas.

Estructura de datos:

```
{
  "id": number,
  "nombre": string,
  "opinion": string,
  "producto": string,
  "calificacion": number
```

}

Proceso de consumo:

1. Try-catch para manejo robusto de errores
2. Fetch asíncrono con sintaxis async/await
3. Almacenamiento en array global testimonios
4. Renderizado con sistema de paginación
5. Generación dinámica de estrellas según calificación

Características implementadas:

- Paginación manual (6 testimonios por página)
- Sistema visual de calificación con íconos
- Carga progresiva de contenido
- Scroll automático suave a nuevos elementos

3.3 Galería de Imágenes (gallery.json)

El archivo gallery.json centraliza todas las imágenes del proyecto con identificadores únicos, rutas relativas y títulos descriptivos para cada imagen.

Estructura de datos:

{

"id": number,

"src": string,

"titulo": string

}

Proceso de consumo:

1. Fetch asíncrono mediante async/await
2. Conversión a objeto JavaScript con .json()
3. Iteración con forEach() sobre array de imágenes
4. Creación dinámica de elementos HTML
5. Inserción en DOM mediante appendChild()

Beneficios del sistema:

- Centralización de recursos de imagen
- Mantenimiento simplificado
- Escalabilidad (agregar imágenes editando solo JSON)
- Carga dinámica y automática
- Atributos de accesibilidad (alt, title)

4. Implementación de jQuery

4.1 Descripción General

La biblioteca jQuery versión 3.6.0 se implementó para proporcionar validaciones en tiempo real del formulario de contacto. La carga se realiza desde Content Delivery Network (CDN) en la sección head del documento HTML (jQuery Foundation, 2021).

CDN utilizado:

https://code.jquery.com/jquery-3.6.0.min.js

Archivo de implementación:

- Ubicación: js/validaciones.js
- Página: index.html
- Sección: Formulario de contacto (#sendEmail)

4.2 Validaciones Implementadas

4.2.1 Validación de Nombre

Utiliza el evento blur para validar cuando el usuario abandona el campo. Verifica longitud mínima de tres caracteres y proporciona feedback visual mediante clases de Bootstrap (is-valid, is-invalid).

Implementación:

```
$('#nombre').on('blur', function() {
    var nombre = $(this).val();
```

```
    if (nombre.length < 3) {
        $(this).removeClass('is-valid').addClass('is-invalid');
    } else {
        $(this).removeClass('is-invalid').addClass('is-valid');
    }
});
```

4.2.2 Validación de Apellido

Implementa validación idéntica al nombre, asegurando consistencia en la experiencia del usuario y longitud mínima de caracteres.

4.2.3 Validación de Correo Electrónico

Utiliza expresión regular para verificar formato válido de correo electrónico, asegurando presencia de símbolo arroba y dominio válido.

Expresión regular utilizada:

```
var emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
```

La validación ejecuta método .test() de JavaScript para verificar coincidencia con el patrón y proporciona feedback visual inmediato.

4.2.4 Contador de Caracteres en Mensaje

Implementa validación en tiempo real mediante evento input. Cuenta caracteres mientras el usuario escribe, crea dinámicamente elemento de visualización si no existe, y valida longitud mínima de diez caracteres.

Funcionalidades:

- Conteo en tiempo real de caracteres
- Creación dinámica de elemento small mediante .after()
- Actualización de texto con .text()
- Validación de longitud mínima

4.2.5 Actualización de Rango de Ingreso

Maneja input de tipo range proporcionando actualización visual en tiempo real del valor seleccionado. Formatea números con separadores de miles utilizando toLocaleString('es-CR').

Implementación:

```
$('#rangoIngreso').on('input', function() {
    var valor = parseInt($(this).val());
    $('#rangoValor').text('¢' + valor.toLocaleString('es-CR'));
});
```

4.2.6 Validación de Grado Académico

Valida select múltiple asegurando que al menos una opción esté seleccionada. Utiliza evento change y verifica longitud del array de valores seleccionados.

4.3 Selectores y Métodos jQuery Utilizados

Selectores implementados:

- $('#id'): Selección por identificador
- $(this): Referencia al elemento actual en contexto de evento

Métodos de manipulación:

- .val(): Obtención y establecimiento de valores
- .addClass(): Adición de clases CSS
- .removeClass(): Remoción de clases CSS
- .text(): Modificación de contenido textual
- .after(): Inserción de elementos hermanos
- .length: Obtención de longitud de colecciones

Métodos de eventos:

- .on('blur', function): Evento al abandonar campo
- .on('input', function): Evento durante escritura
- .on('change', function): Evento al cambiar valor
- $(document).ready(function): Ejecución al cargar DOM

4.4 Ventajas de la Implementación

La utilización de jQuery proporciona sintaxis simplificada comparada con JavaScript vanilla, compatibilidad cross-browser manejada automáticamente por la biblioteca, código más

conciso y legible, feedback inmediato al usuario mejorando experiencia de uso, y capa adicional de validación antes del envío del formulario (Osmani, 2012).

5. Implementación de Multimedia HTML5

5.1 Audio HTML5

El proyecto implementa elemento de audio HTML5 en la sección "Nuestra Historia" del archivo index.html. Utiliza formato MP3 con controles nativos del navegador.

Ubicación:

- Archivo: index.html
- Sección: #about (Nuestra Historia)
- Formato: MP3
- Ruta: media/audio/audio.mp3

Código de implementación:

<audio controls>

<source src="media/audio/audio.mp3" type="audio/mpeg">

Tu navegador no soporta audio.

</source>

Características:

- Controles nativos (reproducción, pausa, volumen, descarga)
- Atributo controls para visibilidad de controles
- Mensaje de fallback para navegadores no compatibles
- Type MIME especificado como "audio/mpeg"

5.2 Video HTML5

El elemento de video se implementa después de la sección de historia, con formato MP4 y controles nativos. Se aplican estilos CSS personalizados para adaptación responsive y presentación visual atractiva.

Ubicación:

- Archivo: index.html
- Sección: #about (después de historia)
- Formato: MP4
- Ruta: media/video/video.mp4

Código de implementación:

```
<video controls class="video-producto">
    <source src="media/video/video.mp4" type="video/mp4">
    Tu navegador no soporta reproducción de video.
</video>
```

Estilos CSS aplicados:

- Ancho: 100% del contenedor padre
- Ancho máximo: 400px (optimizado para formato vertical 3:4)
- Border-radius: 15px para esquinas redondeadas
- Box-shadow para efecto de profundidad

- Centrado automático mediante flexbox

6. Estructura del Proyecto

La arquitectura del proyecto sigue organización modular con separación de responsabilidades entre HTML, CSS, JavaScript y recursos multimedia.

```
proyecto/
├── index.html
├── products.html
├── blog.html
├── imageGallery.html
├── autor.html
├── css/
│   └── style.css
├── js/
│   ├── cart.js
│   ├── gallery.js
│   ├── products.js
│   ├── sendEmail.js
│   ├── testimonials.js
│   ├── ubicacion.js
│   └── validaciones.js
```

```
├── JSON/
│   ├── products.json
│   ├── testimonials.json
│   └── gallery.json
├── img/
│   └── [19 archivos de imagen]
└── media/
    ├── audio/
    │   └── audio.mp3
    └── video/
        └── video.mp4
```

7. Tecnologías Implementadas

7.1 Frontend

- HTML5: Estructura semántica del contenido
- CSS3: Estilos con variables CSS personalizadas
- Bootstrap 5.3.2: Framework responsive
- JavaScript ES6+: Programación moderna con async/await, arrow functions, template literals

7.2 Bibliotecas y Frameworks

- jQuery 3.6.0: Validaciones y manipulación DOM
- Bootstrap Icons: Sistema de iconografía

7.3 APIs y Servicios

- Google Maps JavaScript API: Geolocalización
- FormSubmit: Envío de formularios por correo
- WhatsApp Business API: Integración de carrito

7.4 Almacenamiento

- sessionStorage: Persistencia temporal de carrito

7.5 Herramientas de Desarrollo

- Fetch API: Consumo de recursos JSON
- JSON: Estructura de datos

- Expresiones regulares: Validaciones de formato

8. Métricas del Proyecto

8.1 Código JavaScript Propio

Archivo	Líneas de Código	Funciones	Propósito
cart.js	157	11	Carrito de compras
products.js	98	4	Gestión de productos
testimonials.js	92	6	Sistema de testimonios
sendEmail.js	87	1	Validación de formulario
gallery.js	26	2	Galería de imágenes
ubicacion.js	19	1	Mapa interactivo
validaciones.js	-	6	Validaciones jQuery

Total: 479 líneas de código JavaScript propio

8.2 Recursos JSON

Archivo	Registros	Propósito
products.json	12	Catálogo de productos
testimonials.json	12	Opiniones de clientes
gallery.json	19	Galería de imágenes

Total: 43 registros estructurados

Conclusiones

El proyecto "Sazón de Finca" cumple satisfactoriamente con todos los requerimientos técnicos establecidos para el curso ISW-512. Se implementaron seis archivos JavaScript con código propio totalizando 479 líneas, superando ampliamente el requisito mínimo de tres páginas con JavaScript. El consumo de tres archivos JSON como API REST y la integración de Google Maps API demuestran competencia en el manejo de servicios externos. La implementación de jQuery para validaciones en tiempo real muestra dominio de bibliotecas JavaScript adicionales. Finalmente, la inclusión de elementos multimedia HTML5 de audio y video completa los requerimientos de contenido multimedia del proyecto.

El código desarrollado evidencia aplicación de principios de programación moderna, incluyendo programación asíncrona con async/await, manipulación avanzada del DOM, manejo de eventos, y persistencia de datos con sessionStorage. La arquitectura modular del proyecto facilita el mantenimiento y escalabilidad del sistema.

Referencias

Crockford, D. (2008). *JavaScript: The good parts*. O'Reilly Media.

Flanagan, D. (2020). *JavaScript: The definitive guide* (7th ed.). O'Reilly Media.

González, S. (2024). *ISW-512 Diseño de Aplicaciones Web - Proyecto programado*. Instituto Tecnológico de Costa Rica.

Google Cloud. (2024). *Maps JavaScript API*. https://developers.google.com/maps/documentation/javascript

Google Developers. (2024). *Overview of Google Maps Platform*. https://developers.google.com/maps

Haverbeke, M. (2018). *Eloquent JavaScript: A modern introduction to programming* (3rd ed.). No Starch Press.

jQuery Foundation. (2021). *jQuery API documentation*. https://api.jquery.com/

MDN Web Docs. (2024). *Fetch API*. Mozilla Developer Network. https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

Osmani, A. (2012). *Learning JavaScript design patterns*. O'Reilly Media.

Simpson, K. (2015). *You don't know JS: ES6 & beyond*. O'Reilly Media.