Universidad de Guadalajara

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

Ingeniería Informática.

Actividad 2.3: Funciones asíncronas

Desarrollo de Front End.

Yosef Sánchez Gutiérrez
Marco Alejandro González Mireles
Mirella Stephania Palomera Gómez
Roberto Carlos Martinez Aviña

Mtra. Rosalia Iñiguez.

Introducción

En el desarrollo de aplicaciones web dinámicas, la programación asíncrona es una herramienta esencial que permite ejecutar operaciones complejas como la carga de datos desde servidores sin bloquear la interacción del usuario con la interfaz. JavaScript ofrece varias formas de manejar este tipo de operaciones, incluyendo callbacks, promesas (Promises) y la sintaxis moderna de async/await, que facilitan la escritura de código eficiente, legible y no bloqueante.

A su vez, los eventos del navegador como click, mouseover o change son fundamentales para detectar y responder a las acciones del usuario. Combinados con funciones asíncronas, estos eventos permiten cargar y mostrar información de manera dinámica, mejorando la experiencia en tiempo real. En este contexto, también destaca el uso del operador flecha (=>), que proporciona una forma concisa de escribir funciones, especialmente útil en el manejo de eventos y estructuras como forEach, map o promesas.

En esta práctica se desarrolló una interfaz web simulada de una tienda en línea, en la que se implementaron técnicas de programación asíncrona utilizando Promises, async/await y callbacks anidados, junto con eventos de interacción del usuario. La interfaz incluye botones para cargar productos y usuarios, mostrar mensajes de estado, y limpiar el contenido mostrado. El objetivo principal fue aplicar de forma práctica los conceptos de programación asíncrona en un entorno web interactivo, reforzando el entendimiento de cómo se manejan flujos de datos no bloqueantes en JavaScript.

Desarrollo de la actividad:

El proyecto se divide en tres etapas. La carga de los productos, la carga de los usuarios y el limpiar todo.

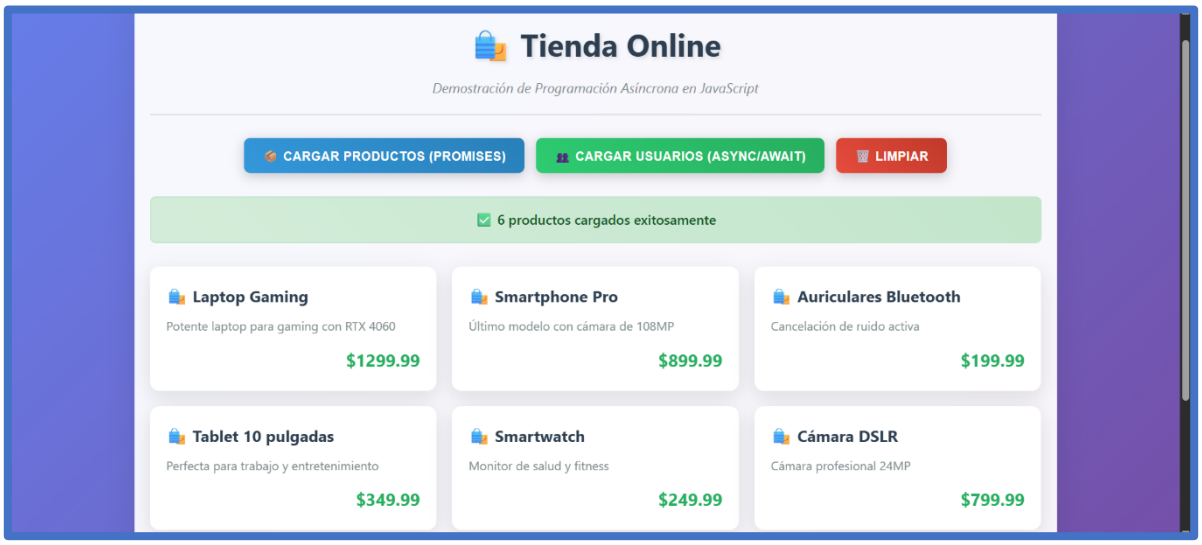
Esta es la página web cuando ingresamos por primera vez:

Cargar productos

Al hacer clic en el botón Cargar Productos (Promises), se ejecuta una función que simula una llamada a servidor utilizando fetchDataFromServer("productos"). Esta devuelve una Promise, la cual es manejada con .then() y .catch() para mostrar los productos o manejar un posible error. Los productos son representados como tarjetas visuales dinámicas que se agregan al DOM.

```
// Función principal para cargar productos usando Promises y Callbacks
const loadProducts = () => {
    loadProductsBtn.disabled = true
    loadProductsBtn.textContent = "⏳ Cargando..."

    // Usar Promise con callbacks
    fetchDataFromServer("productos")
        .then(onProductsLoadSuccess) // Callback de éxito
        .catch(onProductsLoadError) // Callback de error
        .finally(() => {
            // Arrow function para restaurar el botón
            setTimeout(() => {
                loadProductsBtn.disabled = false
                loadProductsBtn.textContent = "📦 Cargar Productos (Promises)"
            }, 500)
        })
}
```

Y así se muestran, incluyendo el mensaje de carga exitosa:

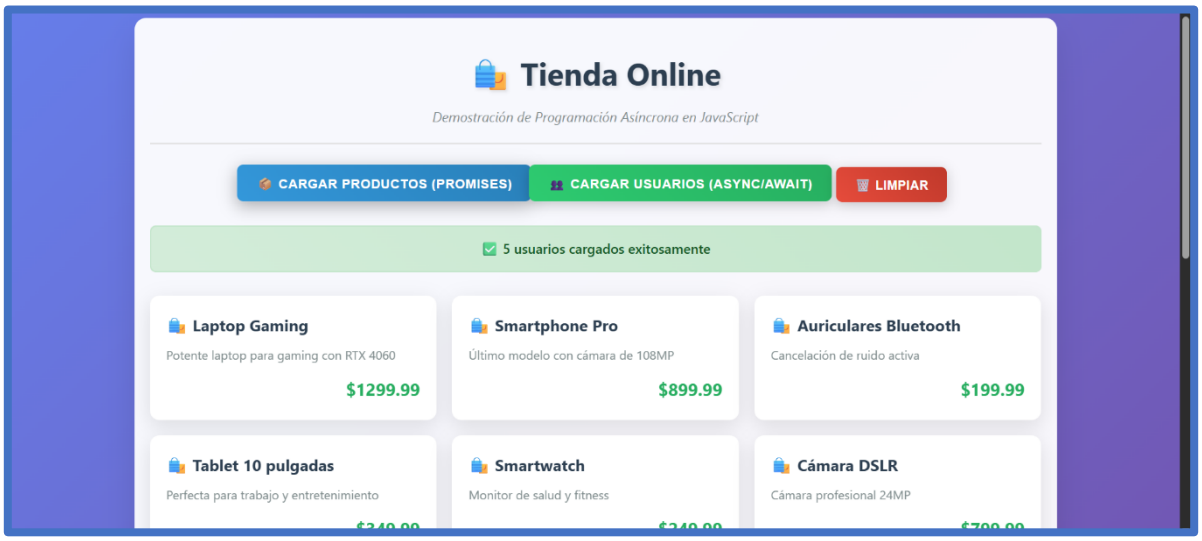


Carga de usuarios

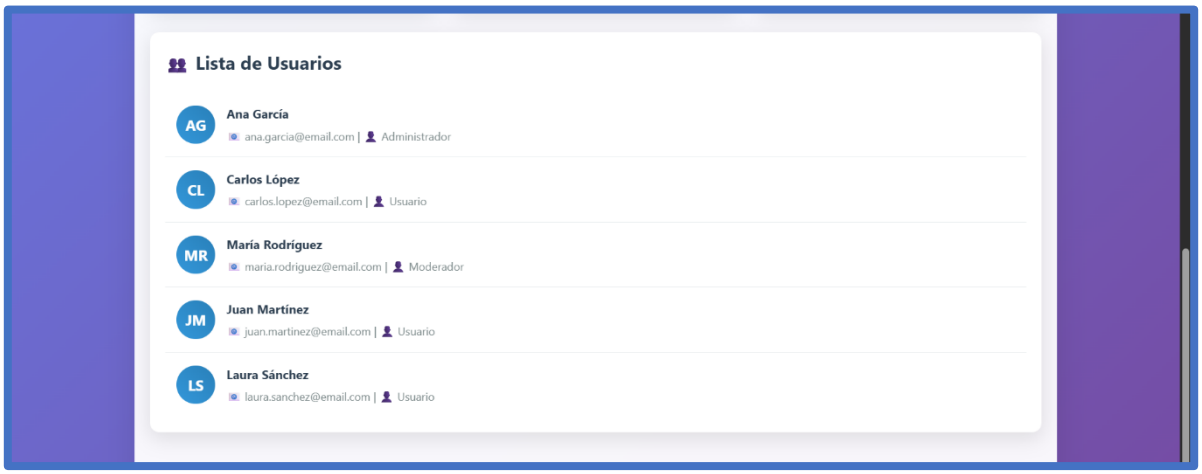
El botón Cargar Usuarios (Async/Await) ejecuta una función declarada con async, donde se utiliza await para esperar la respuesta de fetchDataFromServer("usuarios"). Si los datos son cargados exitosamente, se renderiza una lista con los nombres, correos y roles de los usuarios.

```
// Función async/await para cargar usuarios
const loadUsers = async () => {
    try {
        loadUsersBtn.disabled = true
        loadUsersBtn.textContent = "⏳ Cargando..."

        // Usar async/await
        const users = await fetchDataFromServer("usuarios", 1500)

        hideLoadingSpinner()
        renderUsers(users)
        showStatusMessage(`✅ ${users.length} usuarios cargados exitosamente`, "success")
    } catch (error) {
        hideLoadingSpinner()
        showErrorMessage(`❌ ${error.message}`)
        console.error("Error cargando usuarios:", error)
    } finally {
        // Arrow function para restaurar el botón
        setTimeout(() => {
            loadUsersBtn.disabled = false
            loadUsersBtn.textContent = "👥 Cargar Usuarios (Async/Await)"
        }, 500)
    }
}
```

Primero vemos el mensaje de carga de usuarios exitosa:



Y luego vemos los usuarios ya cargados:



Limpieza de contenido

El botón Limpiar borra el contenido tanto de los productos como de los usuarios y oculta los mensajes de estado. Esta acción también se acompaña de un mensaje confirmando la limpieza.

```
// Función para limpiar el contenido
const clearContent = () => {
    productsGrid.innerHTML = ""
    usersList.innerHTML = ""
    hideMessages()
    hideLoadingSpinner()
    showStatusMessage("Contenido limpiado correctamente", "success")
}
```

Y podemos ver el resultado en la página:

Eventos y animaciones

Todos los botones usan event listeners con funciones flecha para simplificar la sintaxis. Además, se aplican efectos visuales como hover, animaciones fadeInUp para tarjetas y listas, y transiciones suaves para mejorar la experiencia de usuario.

```
195  // Event Listeners usando arrow functions
196  loadProductsBtn.addEventListener("click", () => {
197      console.log("🚀 Iniciando carga de productos...")
198      loadProducts()
199  })
200
201  loadUsersBtn.addEventListener("click", () => {
202      console.log("🚀 Iniciando carga de usuarios...")
203      loadUsers()
204  })
205
206  clearBtn.addEventListener("click", () => {
207      console.log("🧹 Limpiando contenido...")
208      clearContent()
209  })
```

La actividad está compuesta por tres archivos principales:

- **index.html:** Define la estructura general de la página, incluyendo los botones para cargar productos y usuarios, los contenedores para mostrar la información y mensajes de estado.
- **styles.css:** Aplica estilos visuales modernos con gradientes, animaciones y diseño responsivo, mejorando la experiencia de usuario y la presentación de los datos cargados.
- **script.js:** Contiene toda la lógica de programación asíncrona. Implementa funciones que simulan llamadas a servidor usando Promises, async/await y callbacks, además del manejo de eventos del usuario mediante arrow functions y manipulación dinámica del DOM.

Conclusiones:

La práctica realizada permitió comprender y aplicar de manera concreta los conceptos clave de la programación asíncrona en JavaScript, fundamentales para el desarrollo de aplicaciones web modernas e interactivas. A través de la simulación de una tienda en línea, se trabajó con diferentes técnicas como callbacks, promesas y async/await, lo cual facilitó el manejo de operaciones no bloqueantes como la carga simulada de productos y usuarios.

Además, se fortalecieron habilidades relacionadas con la manipulación del DOM, el uso de eventos y la integración de arrow functions para mejorar la legibilidad del código. Al mostrar mensajes de estado, manejar errores y aplicar efectos visuales durante la carga de datos, se resaltó la importancia de ofrecer una experiencia de usuario fluida y reactiva. En conjunto, esta actividad proporcionó una base sólida para comprender cómo funcionan los procesos asíncronos en el navegador y cómo estos pueden ser utilizados de forma práctica en el desarrollo frontend para optimizar la interacción entre el usuario y la aplicación.