

Universidad de Guadalajara

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS



Ingeniería Informática.

Actividad Integradora 2 **AJAX y Fetch API**

Desarrollo de Front End.

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

Mtra. Rosalia Iñiguez.

Introducción:

En el desarrollo moderno de aplicaciones web, la necesidad de obtener datos de servidores remotos sin recargar la página ha dado paso a técnicas como la programación asíncrona con JavaScript. La Fetch API se ha convertido en una herramienta fundamental que permite realizar peticiones HTTP de forma flexible y moderna.

Este proyecto tiene como objetivo desarrollar una aplicación web interactiva que permita obtener datos de usuarios y sus tareas desde una API REST falsa conocida como [JSONPlaceholder](#), empleando programación asíncrona con Fetch y sin recargar la página. La aplicación permite:

- Buscar usuarios por nombre, email o ciudad.
- Consultar tareas de un usuario específico.
- Filtrar tareas por estado: completadas o pendientes.
- Visualizar estadísticas de tareas.
- Utilizar una interfaz web responsive y accesible, que cumpla con buenas prácticas de diseño.

En este documento se describe detalladamente el proceso de diseño, implementación y resultados obtenidos a lo largo de esta actividad.

The screenshot shows a web browser window with the title bar reading "Archivo file:///C:/Users/mirel/Downloads/integradora/index.html". The main content area has a blue header with the text "Gestor de Usuarios y Tareas" and a subtitle "Aplicación con programación asíncrona y Fetch API". Below the header is a search form with a placeholder "Buscar usuario por nombre..." and two buttons: "Buscar" and "Limpiar". A sub-instruction "Escribe el nombre del usuario que deseas encontrar" is displayed below the search input. A table titled "Lista de Usuarios" follows, showing two rows of user data:

ID	Nombre	Correo Electrónico	Ciudad	Acciones
1	Leanne Graham	Sincere@april.biz	Gwenborough	<button>Ver tareas</button>
2	Ervin Howell	Shanna@melissa.tv	Wisokyburgh	<button>Ver tareas</button>

The bottom of the screen shows a Windows taskbar with various icons and system status indicators.

Desarrollo:

Estructura general de la aplicación

El proyecto está organizado en tres archivos principales:

- **index.html:** Define la estructura visual y semántica de la aplicación.
- **styles.css:** Incluye los estilos personalizados aplicando principios de diseño moderno, responsive y accesible.
- **script.js:** Contiene toda la lógica funcional para interacción del usuario, control de estado de la UI y comunicación con la API.

HTML:

El archivo HTML aprovecha las etiquetas semánticas para una estructura clara, modular y accesible. Se integran buenas prácticas como: uso de roles (role="search", aria-label, aria-describedby) para accesibilidad, división lógica de secciones (`<header>`, `<section>`, `<footer>`) con títulos jerárquicos (`<h1>`, `<h2>`), inclusión de formularios que no disparan eventos submit, sino que controlan directamente las acciones con JavaScript.

Componentes:

- **Formulario de búsqueda:** input y botones para buscar o limpiar.
- **Tabla de usuarios:** encabezados bien definidos, con espacio para acciones personalizadas.
- **Vista de tareas:** aparece solo si se elige un usuario, incluye filtros, estadísticas y tabla.

CSS: Estilo y experiencia del usuario

El archivo CSS está cuidadosamente organizado y respeta principios de diseño limpio. Se incluyen características como:

- **Tipografía clara:** uso de fuentes legibles, espacio cómodo y jerarquía visual clara.

- **Feedback visual inmediato:** efectos hover, mensajes de carga, errores estilizados y secciones dinámicas.

Se respetan principios de accesibilidad: enfoque con teclado, elementos identificables, colores no dependientes del estado visual únicamente.

JavaScript:

La lógica funcional se implementa íntegramente en JavaScript moderno, haciendo uso de:

- Fetch API para peticiones HTTP.
- Manipulación del DOM con document.querySelector, createElement, innerHTML.
- Manejadores de eventos (addEventListener) para inputs, botones y filtros.
- Variables globales para guardar el estado actual sin recargar la página.

Carga inicial de usuarios

Al cargar la página se realiza una petición GET:

```
fetch("https://jsonplaceholder.typicode.com/users")
```

Los datos se procesan para mostrar nombre, correo, ciudad y un botón para ver tareas. El proceso contempla:

- Mostrar spinner mientras se carga.
- Ocultar mensajes anteriores.
- Manejar errores con mensajes personalizados y botón de reintento.

Búsqueda de usuarios

Cuando el usuario escribe en el input, se actualiza la tabla sin hacer más peticiones, esto permite una experiencia fluida e inmediata. Se manejan también los eventos de teclado (Enter) y botones.

The screenshot shows a web application interface. At the top, a blue header bar displays the title "Gestor de Usuarios y Tareas" and a subtitle "Aplicación con programación asíncrona y Fetch API". Below the header, a search form has a placeholder "Buscar usuario por nombre..." and two buttons: "Buscar" (blue) and "Limpiar" (grey). A text input field below the search form contains the placeholder "Escribe el nombre del usuario que deseas encontrar". A table follows, with columns: ID, Nombre, Correo Electrónico, Ciudad, and Acciones. Two rows are shown:

ID	Nombre	Correo Electrónico	Ciudad	Acciones
1	Leanne Graham	Sincere@april.biz	Gwenborough	<button>Ver tareas</button>
2	Ervin Howell	Shanna@melissa.tv	Wisokyburgh	<button>Ver tareas</button>

At the bottom of the main content area, there is a section titled "Tareas de Kurtis Weissnat" which is currently empty. The browser's status bar at the bottom right shows the time as 11:11 p.m. and the date as 26/06/2025.

Visualización de tareas

Al dar clic en "Ver tareas", se obtiene información personalizada del usuario:

```
fetch(`https://jsonplaceholder.typicode.com/todos?userId=${userId}`)
```

La tabla muestra:

- ID de la tarea
- Título descriptivo
- Estado visualizado como o según si está completada.

La sección se revela dinámicamente con scroll suave y carga progresiva.

The screenshot shows a web-based application for managing tasks. At the top, there's a header bar with a back arrow, forward arrow, file menu, and download options. A dropdown menu labeled 'Filtrar por estado:' is set to 'Todas las tareas'. Below the header is a summary section with three large blue boxes: 'Total' (20), 'Completadas' (9), and 'Pendientes' (11). Underneath is a table with columns for 'ID', 'Título de la Tarea', and 'Estado'. The table lists seven tasks, each with a status indicator (green checkmark for completed, red X for pending) and a timestamp in the bottom right corner.

ID	Título de la Tarea	Estado
121	inventore aut nihil minima laudantium hic qui omnis	Completada
122	provident aut nobis culpa	Completada
123	esse et quis iste est earum aut impedit	Pendiente
124	qui consectetur id	Pendiente
125	aut quasi autem iste tempore illum possimus	Pendiente
126	ut asperiores perspiciatis veniam ipsum rerum saepe	Completada
127	voluptatem libero consectetur rerum ut	Completada

Filtro de tareas por estado

Se incluye un <select> que permite mostrar:

- Todas las tareas
- Solo completadas
- Solo pendientes

La lógica filtra en memoria y no vuelve a hacer fetch.

The screenshot shows a web-based application titled "Tareas de Leanne Graham". At the top, there is a filter dropdown labeled "Filtrar por estado:" with "Todas las tareas" selected. A tooltip for this dropdown says "Filtrá las tareas según su estado de completado". Below the filter are three blue cards: "Total" (20), "Completadas" (11), and "Pendientes" (9). A modal window is open over the filter, showing three options: "Todas las tareas" (selected), "Solo completadas", and "Solo pendientes". The main area displays a table of tasks:

ID	Título de la Tarea	Estado
1	delectus aut autem	Pendiente
2	quis ut nam facilis et officia qui	Pendiente
3	fugiat veniam minus	Pendiente
4	et porro tempora	Completada
5	laboriosam mollitia et enim quasi adipisci quia provident illum	Pendiente

Estadísticas visuales

Además de mostrar tareas, se agregan contadores visuales con colores y estilo de tarjetas, esto permite tener un resumen rápido y claro de la carga de trabajo.

Gestión de errores y estados

Cada fetch incluye catch para manejar problemas de conexión. Se incluye un mensaje visible con ícono, texto explicativo y botón para reintentar.

Además, cuando no hay coincidencias en la búsqueda, se muestra un mensaje visual con ícono amigable.

Conclusiones:

Este proyecto fue un ejercicio integral que nos permitió profundizar en diversas áreas fundamentales del desarrollo frontend, especialmente la comunicación asincrónica con APIs y la creación de interfaces dinámicas.

Durante el desarrollo se reforzaron habilidades como:

- Escritura de código limpio y modular.
- Separación de responsabilidades entre HTML, CSS y JS.
- Manejo de eventos en formularios y elementos interactivos.
- Interpretación de estructuras de datos complejas (JSON anidados).
- Comunicación asincrónica y gestión de errores de red.

Además, se integraron aspectos clave de experiencia del usuario, como tiempos de respuesta claros, retroalimentación inmediata y una interfaz responsive accesible desde cualquier dispositivo.

Entre los principales aprendizajes se encuentra la importancia de anticipar los posibles estados de la aplicación (cargando, error, sin resultados), y proveer soluciones visuales claras para cada uno.

Esta actividad no solo contribuyó al desarrollo de habilidades técnicas, sino que también reforzó la planificación y organización de una aplicación de principio a fin, simulando escenarios reales de trabajo en entornos web.