Universidad de Guadalajara

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS



Ingeniería Informática.

Actividad 3.3: AJAX

Desarrollo de Front End.

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

Mtra. Rosalia Iñiguez.

Introducción

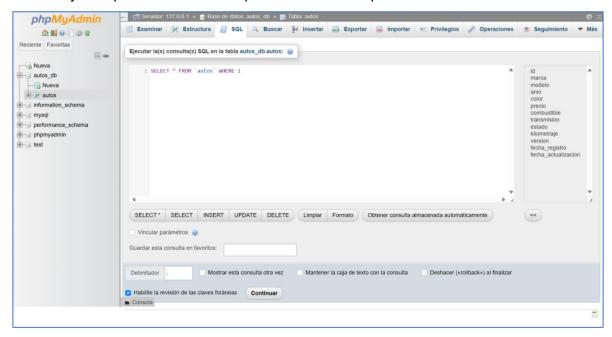
En esta actividad desarrollamos un sistema web que utiliza transferencias asíncronas para mostrar información almacenada en una base de datos MySQL a través de PHP y AJAX. A diferencia de la actividad anterior, donde los datos provenían de un archivo XML, en esta ocasión la información se consulta directamente desde el servidor mediante el método ajax() de la librería jQuery, logrando así una comunicación eficiente y dinámica entre el cliente y el servidor.

El objetivo principal fue crear una tabla interactiva capaz de mostrar el total de registros, incluir paginación, opciones de búsqueda y ordenamiento ascendente o descendente, y permitir agregar, modificar o eliminar autos del sistema, todo sin necesidad de recargar completamente la página. Esto reduce la carga del servidor y mejora la experiencia del usuario, ya que solamente se actualizan las secciones necesarias de la interfaz.

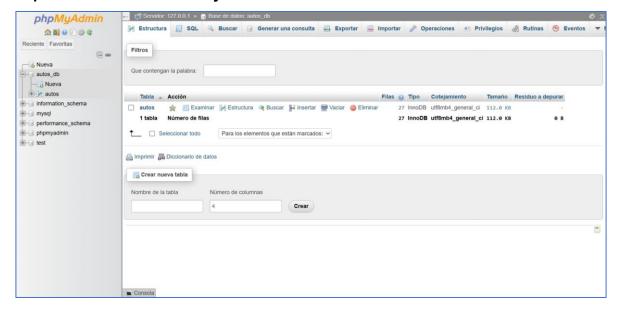
A lo largo del proceso aprendimos a implementar consultas AJAX GET y POST, crear peticiones para filtrar, ordenar o paginar los registros y desarrollar formularios que envían datos de forma asíncrona para agregar, editar o eliminar elementos en la base de datos. También reforzamos el uso de sesiones y tokens de seguridad en PHP para controlar el acceso a la información, así como la integración de jQuery en proyectos con múltiples funcionalidades y eventos.

Vista de la Base de Datos en phpMyAdmin

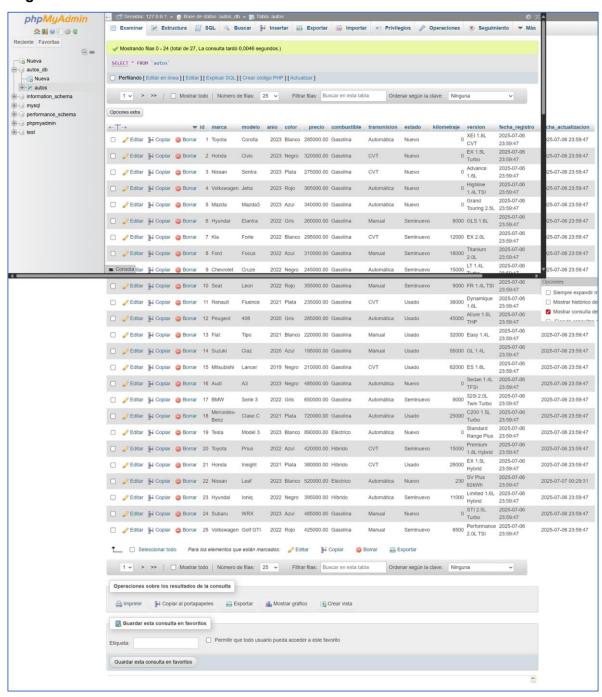
La siguiente captura muestra la estructura de la base de datos utilizada para este proyecto. Se creó en MySQL local y contiene la tabla que almacena los autos con sus respectivos atributos como ID, marca, modelo, año, color, precio, combustible, transmisión, estado, kilometraje y versión. Esta base de datos es la fuente principal de información que se consulta y manipula a través de las peticiones AJAX implementadas en el sistema.



Captura de la base de datos y su tabla autos



Registros de la actividad



Este código realiza una petición AJAX de tipo GET al script PHP listar_auto.php, enviando como parámetros los filtros de búsqueda, la cantidad de registros a mostrar, el orden y la página actual. Al recibir la respuesta en formato JSON, valida si fue exitosa, muestra los autos en la tabla y actualiza las estadísticas de la interfaz. Si ocurre un error, despliega un mensaje informativo al usuario.

```
url: "api/listar_auto.php",
type: "GET",
 search: search,
 limit: cantidad,
 orderBy: orden,
 orderDir: direccion,
 page: pagina,
dataType: "json",
 console.log("☑ Respuesta recibida:", response)
   const autos = response.data || []
   const total = response.total || 0
   actualizarEstadisticas(total, cantidad)
   if (autos.length === 0) {
     showMessage("No se encontraron registros", "info")
   showMessage("Error al cargar datos: " + (response.message || "Error desconocido"), "error")
 console.error("X Error AJAX:", error)
```

Captura del test de la conexión de la bd con el backend en php:

Test de Conexión

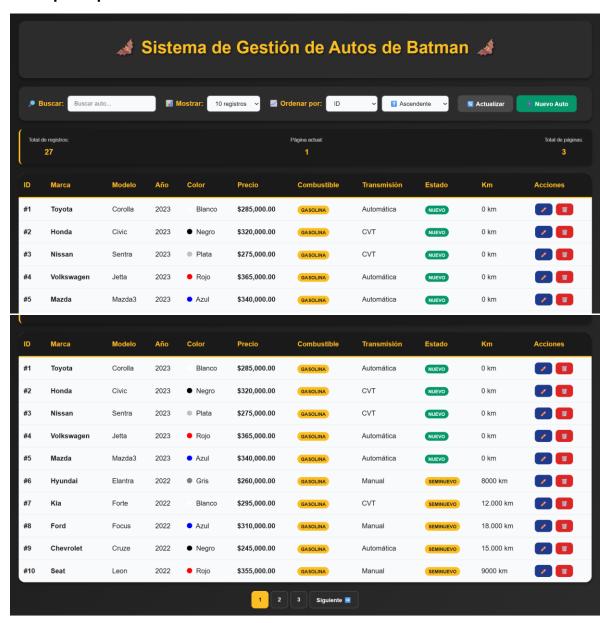
Conexión exitosa

✓ Total de autos en BD: 27

Primeros 5 autos:

- Toyota Corolla (2023)
- Honda Civic (2023)
- Nissan Sentra (2023)
- Volkswagen Jetta (2023)
- Mazda Mazda3 (2023)

Vista principal



Filtrados

El filtrado por campo se realiza principalmente en la función cargarAutos() con este fragmento

```
// Función principal para cargar autos
function cargarAutos() {
   console.log(" Cargando autos...")

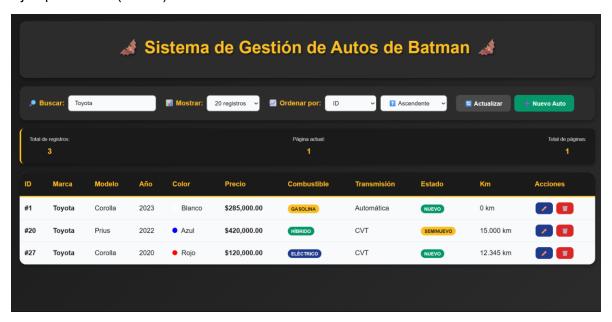
const search = $("#buscar").val()
```

Aquí se obtiene el valor que el usuario escribe en el input de búsqueda con id buscar.

Luego, ese valor se envía como parámetro en la petición AJAX:

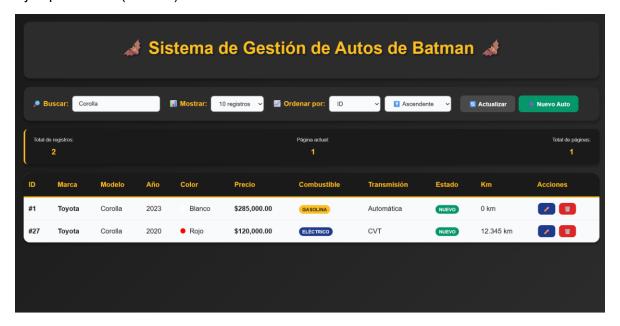
```
$.ajax({{
    url: "api/listar_auto.php",
  type: "GET",
 data: {
   search: search,
   limit: cantidad,
   orderBy: orden,
   orderDir: direccion,
   page: pagina,
 dataType: "json",
   console.log("☑ Respuesta recibida:", response)
    if (response.success !== false) {
     const autos = response.data || []
      const total = response.total | | 0
      actualizarEstadisticas(total, cantidad)
      mostrarPaginacion(total, cantidad)
      if (autos.length === 0) {
        showMessage("No se encontraron registros", "info")
      showMessage("Error al cargar datos: " + (response.message || "Error desconocido"), "error")
   console.error("X Error AJAX:", error)
```

Ejemplo filtrado (marca)





Ejemplo filtrado (modelo)

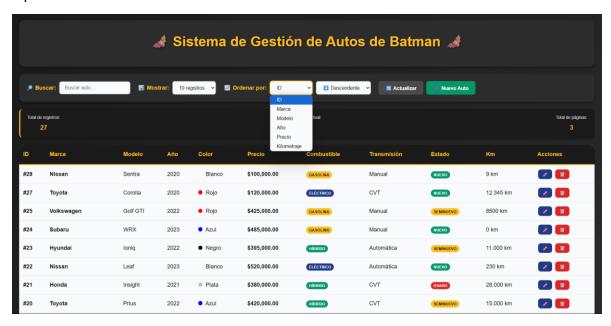


Ordenamiento

El ordenamiento se implementa de forma asíncrona utilizando los campos de selección en la interfaz. Cuando el usuario cambia la columna o la dirección de orden, se ejecuta de nuevo la función cargarAutos(), enviando los parámetros orderBy y orderDir al servidor mediante AJAX. PHP usa estos valores para construir la consulta SQL ordenada antes de devolver los resultados filtrados y ordenados en formato JSON, los cuales se muestran en la tabla sin recargar la página.

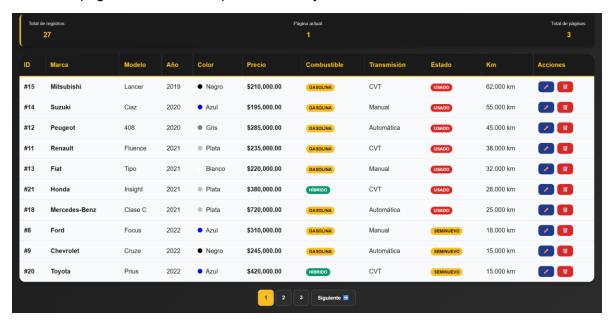


Opciones de ordenamiento





Probando paginación ordenada por kilomentraje en orden descendente:



CRUD

El sistema implementa las cuatro operaciones básicas del CRUD de manera asíncrona mediante AJAX:

- Crear: Agrega nuevos autos con POST.
- Leer: Lista autos aplicando filtros, búsqueda y paginación con GET.
- Actualizar: Edita autos existentes mediante POST.
- Eliminar: Borra autos mediante POST.

Todas estas operaciones se realizan sin recargar la página completa, mejorando la experiencia de usuario y reduciendo la carga en el servidor.

```
$.ajax({
url: url,
type: "POST",
data: formData,
dataType: "json",
success: (response) => {
    if (response.success !== false) {
        showMessage("Auto ${accion} exitosamente! ..., "success")
        cerrarModal()
        cargarAutos()
    } else {
        showMessage("Error al guardar: " + (response.message || response.error || "Error desconocido"), "error")
    }
},
error: (xhr, status, error) => {
        console.error("Error al guardar:", error)
        showMessage("Error de conexión al guardar", "error")
},
})
```

Editar Auto

```
function editar(id) {
    $.ajax({
        url: "api/listar_auto.php",
        type: "GET",
        data: { id: id },
        dataType: "json",
        success: (response) => {
        if (response.data && response.data.length > 0) {
            abrirModal("editar", response.data[0])
        } else {
            showMessage("No se pudo cargar la información del auto", "error")
        },
        error: () => {
            showMessage("Error al cargar datos para editar", "error")
        },
    })
}
```

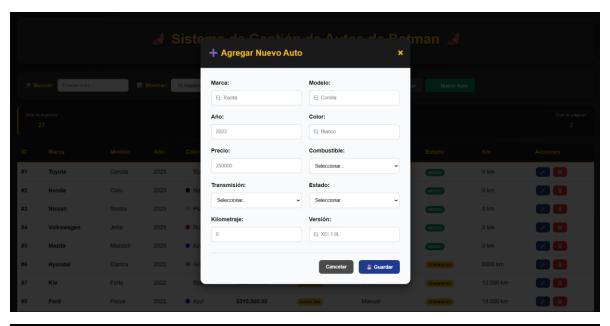
Eliminar auto

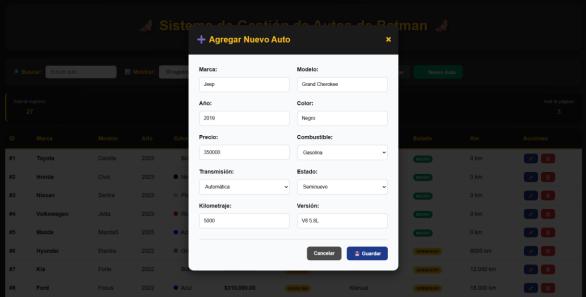
```
// Eliminar auto
function eliminar(id) {
    if (!confirm("¿Estás seguro de que deseas eliminar este auto?\n\nEsta acción no se puede deshacer.")) {
        return
    }

$.ajax({
        url: "api/eliminar_auto.php",
        type: "POST",
        data: { id: id },
        data: { id: id },
        datarype: "json",
        success: (response) => {
        if (response.success !== false) {
            showMessage("Auto eliminado exitosamente! \overline{w}", "success")
            cargarAutos()
        } else {
            showMessage("Error al eliminar: " + (response.message || response.error || "Error desconocido"), "error")
        }
    },
    error: (xhr, status, error) => {
        console.error("Error al eliminar:", error)
        showMessage("Error de conexión al eliminar", "error")
    }
}
```

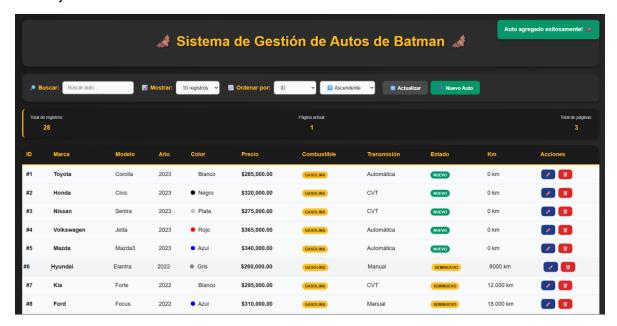
Vista del CRUD en la página

Crear





Mensaje de confirmación

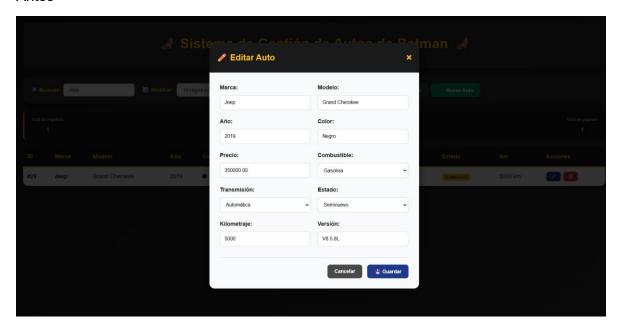


Verificamos la creación

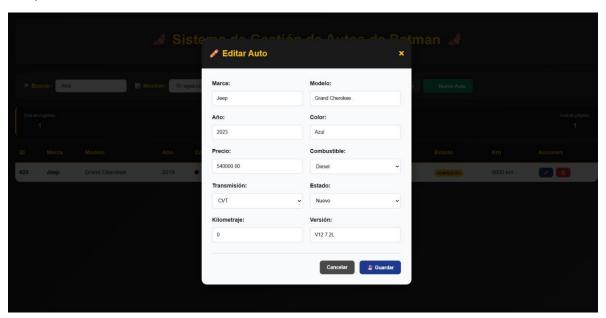


Editar auto

Antes



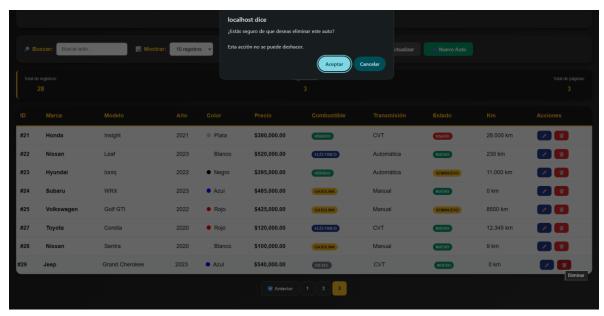
Después



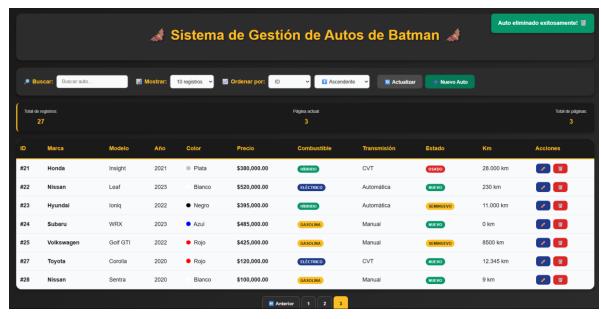
Mensaje de verificación



Eliminar registro



Mensaje de verificación





Conclusiones:

La práctica realizada permitió comprender y aplicar de manera concreta los conceptos clave de las peticiones asíncronas mediante AJAX, fundamentales para el desarrollo de aplicaciones web dinámicas e interactivas. A través de la creación de un sistema de gestión de autos, se implementaron operaciones CRUD completas (crear, leer, actualizar y eliminar) integradas con PHP y MySQL, permitiendo la comunicación entre el cliente y el servidor sin recargar la página. Además, se fortalecieron habilidades relacionadas con el uso de jQuery para realizar solicitudes GET y POST, el manejo de respuestas JSON y la actualización dinámica del DOM para mostrar resultados en tiempo real. Se evidenció la importancia de implementar paginación, filtros de búsqueda, ordenamiento y mensajes de estado para brindar al usuario una experiencia fluida y eficiente al interactuar con los datos.

En conjunto, esta actividad proporcionó una base sólida para comprender cómo funcionan los procesos asíncronos en aplicaciones web modernas y cómo su correcta implementación puede optimizar la interacción, el rendimiento y la escalabilidad de los sistemas frontend conectados a bases de datos remotas.