INFORME PROYECTO

AUTOMARKET FULL STACK

1. VISIÓN GENERAL DEL PROYECTO

AutoMarket es una aplicación web dinámica diseñada para la gestión y visualización de un catálogo de vehículos. El objetivo principal fue retratar la experiencia de una concesionaria tradicional, permitiendo la actualización de precios, stock y detalles técnicos en tiempo real a través de una base de datos, eliminando la dependencia de archivos HTML estáticos.

2. FUNCIONALIDADES PRINCIPALES DEL SISTEMA

Catálogo de Inventario en Tiempo Real: El sistema carga los vehículos dinámicamente desde la base de datos al iniciar la aplicación. Incluye una lógica de Paginación con un botón de Ver Más que solicita datos en bloques para optimizar el rendimiento, y una función inteligente que detecta el final de la lista para cambiar el botón a Ver Menos.

Motor de Búsqueda y Filtrado Completo: El usuario puede refinar su búsqueda mediante múltiples criterios simultáneos. Incluye una búsqueda por texto libre (Marca o Modelo), filtros numéricos de rango de precios (Mínimo y Máximo), filtros de categoría por Marcas específicas y un sistema de ordenamiento por Precio, Kilometraje y Año.

Experiencia de Usuario y Ventana Modal: Al seleccionar un vehículo, se despliega una ventana emergente sobre la interfaz. Esta muestra la foto ampliada, una ficha técnica con íconos visuales y un botón de consulta para la compra directa. Esto mejora la navegación al evitar recargas de página innecesarias.

Diseño 100 por ciento Responsive: La interfaz se adapta automáticamente a cualquier dispositivo móvil. La grilla de productos cambia de 3 columnas en escritorio a 1 columna en móvil, y los menús y filtros se reestructuran para una navegación táctil fluida.

Página de Login y Publicar auto: Además de la landing principal el sistema cuenta al inicio de la página con un sistema de login y ya dentro de la página con un botón para publicar su auto,este le envía a un formulario a completar.

3. ARQUITECTURA DEL SISTEMA

El proyecto utiliza una arquitectura de 3 Capas, separando claramente las responsabilidades para facilitar el mantenimiento y la escalabilidad.

Capa de Base de Datos :

Tecnología: SQL Server Express.

Función: Es el repositorio de los autos. Almacena la información de forma permanente y segura.

Capa de Logica y API (Backend):

Tecnología: ASP.NET Core Web API(Net 8) en C# a través de Visual Studio.

Función: Actúa como intermediario y cerebro. Recibe las peticiones de la web, aplica la lógica de negocio (filtros, ordenamiento) y se comunica con la base de datos.

Capa de Frontend:

Tecnología: HTML, CSS y JavaScript.

Función: Es la interfaz visual, JavaScript Se encarga de pedir los datos a la API para conectarlos con HTML y CSS para luego mostrarlos en la pantalla del usuario.

3.1. Flujo de Comunicación Cliente-Servidor:

La comunicación entre el Frontend y el Backend se realiza mediante el protocolo HTTP, siguiendo el patrón de arquitectura REST.

Petición (Request): El Frontend (`script.js`) utiliza la función `fetch()` para enviar una solicitud HTTP GET al endpoint de la API (`/api/vehiculos`).

Procesamiento: El Backend (.NET) recibe la solicitud en el Controlador, consulta la base de datos SQL Server mediante Entity Framework y recupera la información.

Respuesta (Response): La API serializa los objetos C# y devuelve los datos en formato JSON (JavaScript Object Notation).

Renderizado: El navegador recibe el JSON y JavaScript actualiza dinámicamente el DOM (HTML) para mostrar las tarjetas de vehículos al usuario sin recargar la página.

4-DETALLES DE IMPLEMENTACIÓN TÉCNICA

4.1. BASE DE DATOS (SQL SERVER)

La Estructura de la Tabla Vehículos: La tabla principal Vehiculos fue diseñada con tipos de datos específicos para garantizar la integridad de la información.

Columna Id: (INT, IDENTITY), Es la clave primaria. Se configuró como autoincremental (Identity 1,1) para que la base de datos generará un identificador único automáticamente con cada nuevo auto, evitando duplicados.

Columna Precio: (DECIMAL 18,2), Se eligió el tipo decimal para manejar valores monetarios con precisión exacta, evitando errores de redondeo financiero.

Columna ImagenUrl: (VARCHAR), Por eficiencia, no guardamos la imagen física en la base de datos. Guardamos la ruta o nombre del archivo (ej: gol.png), manteniendo la base de datos ligera y rápida.

Columnas de Texto:(Marca, Modelo, Descripción), Utilizan VARCHAR para permitir almacenamiento de texto.

Columna Transmisión: Para diferenciar entre las marchas de los vehículos**.** (Manual/Automática).

Conexión y Seguridad: La conexión se realiza mediante Windows Authentication (Trusted Connection), eliminando la necesidad de guardar contraseñas de base de datos en el código fuente durante el desarrollo.

4.2. BACKEND (.NET C#)

El motor del sistema está construido sobre ASP.NET Core.

Entity Framework Core (ORM): Implementamos un Mapeador Objeto-Relacional para no escribir SQL manual.

Clase Contexto (AutoMarketContext): Hereda de DbContext y gestiona la sesión con la base de datos.

DbSet Vehiculos: Representa la tabla en memoria. Cada operación sobre esta lista (como .ToList o .Where) es traducida automáticamente por EF Core a una consulta SQL.

Controlador (VehiculosController): Expone un Endpoint principal GET api/vehiculos que acepta múltiples parámetros opcionales (buscar, min, max, marca, orden, pagina). Uso de LINQ: Toda la lógica de filtrado se construyó utilizando

Language Integrated Query. Esto permite construir la consulta dinámicamente paso a paso (agregando Where según los filtros activos) antes de ir a la base de datos, lo cual es altamente eficiente.

Inyección de Dependencias: Configuramos el contenedor de servicios en Program.cs para inyectar el contexto de datos en el controlador, asegurando un manejo eficiente de las conexiones a la base de datos.

4.3. FRONTEND WEB

La interfaz visual se construyó siguiendo el principio de Separación de Intereses.

JavaScript (script.js): Es el controlador del lado del cliente. Utiliza la Fetch API para realizar peticiones asíncronas (AJAX) al backend.

Manejo del Estado: Utiliza variables globales para recordar la página actual, los filtros aplicados y el término de búsqueda, permitiendo una navegación fluida sin perder el contexto.

Manipulación del DOM: Genera el HTML de las tarjetas dinámicamente usando Template Strings, inyectando los datos recibidos (Precio, Foto, Info) en el contenedor principal.

Diseño (CSS): Se implementó CSS Grid para la grilla de productos, permitiendo un diseño flexible y alineado. Se utilizaron Media Queries para transformar el layout en dispositivos móviles, pasando de una fila horizontal a una columna vertical apilada

5. HISTORIA DEL DESARROLLO (PASO A PASO)

El proyecto se construyó siguiendo esta metodología:

Fase 1: Comenzamos creando la base de datos en SQL Server y poblando la tabla con datos de prueba reales. Definimos el modelo de datos en CSharp para que coincida exactamente con la tabla SQL.

Fase 2: Configuramos el proyecto ASP.NET Core y conectamos Entity Framework. Creamos el controlador básico para devolver la lista de autos y probamos la conexión utilizando Swagger, asegurando que los datos fluyan desde el SQL hasta el formato JSON.

Fase 3: Conectamos la página web existente con la API. Reemplazamos el contenido estático HTML por un contenedor vacío y escribimos la lógica en JavaScript para llenar ese contenedor con los datos reales provenientes del Backend. Además solucionamos problemas de seguridad CORS para permitir la comunicación entre el navegador y el servidor.

Fase 4:Implementamos el buscador por texto, los filtros de rango de precios y el ordenamiento. Esto requirió modificar tanto el Backend (para procesar los filtros en SQL) como el Frontend (para capturar los eventos de usuario y enviar los parámetros correctos).

Fase 5: Finalizamos con las mejoras de experiencia de usuario como la ventana modal para ver detalles sin salir de la página, la paginación inteligente con el botón Ver Más Y la adaptación completa a dispositivos móviles (Responsive Design).

6. ESTADO ACTUAL

El proyecto se encuentra FINALIZADO en su versión MVP (Producto Mínimo Viable). Todos los sistemas (Base de Datos, API y Web) están integrados y funcionando en conjunto. El código es estable, modular y está listo para ser escalado con nuevas funcionalidades.

7. FUTURAS MEJORAS

Para una siguiente etapa de evolución comercial, se proponen:

1. **Panel de Administración privado**:Para cargar y editar autos sin tocar la base de datos.
2. **Sistema de autenticación real** con usuarios y contraseñas encriptadas.