



INSTITUTO CEI INFORME OBLIGATORIO 2023 AUTOMOTORA

ANALISTA
PROGRAMADOR
PROGRAMACIÓN II

Diciembre 2023

Profesor Gonzalo Duarte

Federico Goldaracena
Nicolás Olivera

Resumen.

En esta propuesta de aplicación web, se ha diseñado una simulación detallada que emula el funcionamiento de una entidad automotriz. El desarrollo se lleva a cabo mediante el uso de C# y ASP.NET, garantizando así una implementación robusta y eficiente. La gestión integral de la entidad abarca diversas áreas clave, entre las que se encuentran clientes, vehículos, usuarios, ventas, alquileres y bases de datos. La información correspondiente a estas áreas se organiza y gestiona mediante el empleo de listas, proporcionando una estructura eficaz y dinámica.

Cuando un usuario accede a la aplicación, se encuentra con una interfaz de inicio de sesión que actúa como el punto de entrada al sistema. La asignación de permisos se lleva a cabo de manera inteligente, ajustándose a las necesidades y responsabilidades específicas de cada tipo de usuario. Un ejemplo claro de esta diferenciación se observa entre el administrador y el vendedor.

El administrador, como figura clave en la gestión, goza de amplias facultades que abarcan la visualización y administración de usuarios, clientes, vehículos, ventas y alquileres. Su capacidad de acción incluye la realización de altas, bajas, modificaciones, actualizaciones y eliminaciones de registros en todas estas categorías. Este nivel de control y autoridad permite al administrador desempeñar un papel central en la supervisión y dirección de la entidad automotriz simulada.

En contraste, el vendedor cuenta con permisos más específicos y orientados a su función principal. Su acceso se centra en la consulta y ejecución de ventas, lo que optimiza su experiencia dentro de la aplicación y garantiza un flujo de trabajo eficiente y enfocado en sus responsabilidades directas.

En resumen, esta aplicación web no solo simula la operación de una entidad automotriz, sino que también demuestra una estructura de gestión versátil y adaptativa, garantizando que cada usuario tenga acceso solo a las funciones necesarias para su rol específico. La implementación de tecnologías como C# y ASP.NET contribuye a la solidez y eficacia del sistema en su conjunto.

Contenido

1. Introducción	1
1.1 .NET	1
1.2 ASP.NET	1
1.3 Visual Studio	2
1.4 C#	2
1.4.1 Listas	2
1.5 UML	2
2. Desarrollo	3
2.1 UML	3
2.2 Clases	4
2.2.1 Clase Vehiculo	4
2.2.2 Clases Hijas	5
2.2.3 Clase Usuario	5
2.2.4 Clase Cliente	6
2.2.5 Clase Venta	6
2.2.6 Clase Alquiler	7
2.2.7 Clase BaseDeDatos	8
2.3 Site.Master	10
2.4 Vehiculos	11
2.5 Usuarios	16
2.6 Clientes	17
2.7 Ventas	17
2.8 Alquileres	18
3. Resultados	18
4. Conclusiones	23
5. Referencias	24

1. Introducción

1.1 .NET

.NET es una plataforma de código abierto que facilita la creación de aplicaciones para escritorio, web y móviles, con capacidad de ejecutarse en diversos sistemas operativos. La plataforma abarca herramientas, bibliotecas y lenguajes que respaldan el desarrollo de software moderno, escalable y de alto rendimiento. Respaldada por una comunidad de desarrolladores activa, .NET se mantiene y evoluciona constantemente. En resumen, .NET realiza varias funciones esenciales: **Compilación de Código:** Traduce el código del lenguaje de programación .NET en instrucciones comprensibles para los dispositivos de computación. **Utilidades para Desarrollo Eficiente:** Ofrece utilidades que facilitan el desarrollo de software eficiente, como la obtención de la hora actual o la impresión de texto en pantalla. **Gestión de Tipos de Datos:** Define un conjunto de tipos de datos para almacenar información, como texto, números y fechas en los equipos. Esta funcionalidad es crucial para la manipulación y almacenamiento de datos en las aplicaciones desarrolladas con .NET. [1]

1.2 ASP.NET

Es un marco web gratuito que permite crear sitios web y aplicaciones web utilizando HTML, CSS y JavaScript. También posibilita la creación de API web y la integración de tecnologías en tiempo real, como Web Sockets. ASP.NET Core es una alternativa a ASP.NET, y se brinda orientación sobre cómo elegir entre ambas. Para comenzar, se recomienda instalar Visual Studio Community Edition, un entorno de desarrollo gratuito para ASP.NET en sistemas Windows. Existen tres marcos principales dentro de ASP.NET para crear aplicaciones web: Web Forms, ASP.NET MVC y ASP.NET Web Pages. Cada uno tiene su propio estilo de desarrollo y se adapta a diferentes niveles de experiencia y preferencias de programación. **Web Forms:** Ideal para aquellos con experiencia en Win Forms, WPF, o .NET. Ofrece un desarrollo rápido mediante una biblioteca de controles que encapsulan el marcado HTML. **MVC:** Recomendado para quienes tienen experiencia en Ruby on Rails o .NET. Proporciona control total sobre el marcado HTML, separación clara entre código y marcado, y es adecuado para aplicaciones móviles y de página única (SPA). **Web Pages:** Indicado para aquellos familiarizados con ASP clásico o PHP. Permite combinar código de servidor con HTML de manera rápida y ligera. Es importante destacar que los tres marcos comparten la funcionalidad principal de .NET y ASP.NET, como el modelo de seguridad de inicio de sesión. Además, no son mutuamente excluyentes y se pueden usar en la misma aplicación web. Además de estos marcos, ASP.NET ofrece opciones para la creación de API web con ASP.NET Web API y la implementación de funcionalidades en tiempo real con ASP.NET SignalR. En cuanto al desarrollo de sitios y aplicaciones móviles, ASP.NET puede potenciar aplicaciones nativas con un back-end de API web y sitios web móviles mediante marcos de diseño como Twitter Bootstrap. También se destaca la capacidad de desarrollar aplicaciones de página única (SPA) con HTML5, CSS3 y JavaScript, con plantillas disponibles en Visual Studio. ASP.NET proporciona una variedad de opciones y herramientas para el desarrollo web, adaptándose a diferentes necesidades y niveles de experiencia. [2]

1.3 Visual Studio

Visual Studio destaca como una poderosa herramienta de desarrollo que facilita la gestión integral del ciclo de desarrollo en un único entorno. Este entorno de desarrollo integrado (IDE) ofrece capacidades completas para escribir, editar, depurar y compilar código, además de permitir la implementación de la aplicación resultante. Más allá de las funciones básicas de edición y depuración de código, Visual Studio proporciona una gama de herramientas y características adicionales.

Entre ellas se incluyen compiladores, utilidades de finalización de código, control de código fuente, extensiones y diversas funcionalidades diseñadas para optimizar cada etapa del proceso de desarrollo de software. En síntesis, Visual Studio se presenta como una solución integral que va más allá de la simple edición de código, brindando a los desarrolladores un conjunto completo de recursos para mejorar la eficiencia y la calidad en todas las fases del desarrollo de aplicaciones. [3]

1.4 C#

C# (pronunciado C Sharp) representa una evolución significativa realizada por Microsoft, amalgamando lo más destacado de los lenguajes C y C++. A lo largo de su desarrollo continuo, se le han incorporado funcionalidades provenientes de otros lenguajes, como Java, aprovechando aspectos de su sintaxis evolucionada. Este lenguaje, orientado a objetos en toda la plataforma NET (tanto Framework como Core), ha ido adquiriendo las facilidades de creación de código presentes en Visual Basic, otro de los lenguajes eminentemente utilizado por Microsoft. Esta amalgama le confiere a C# una versatilidad excepcional, convirtiéndolo en un lenguaje accesible y fácil de aprender, sin sacrificar la potencia inherente a C. Con la llegada de la versión .NET Core, se ha llevado a cabo una reconstrucción completa del compilador de C#, resultando en una mejora impresionante: las aplicaciones ahora se ejecutan hasta un 600% más rápido que en versiones anteriores. Este avance sustancial refuerza la posición de C# como un lenguaje moderno, potente y eficiente dentro del ecosistema de desarrollo de Microsoft. [4]

1.4.1 Listas

En el contexto de C#, una lista (List) representa una estructura de datos diseñada para almacenar una colección de elementos del mismo tipo en un orden secuencial. A diferencia de los arrays, las listas carecen de un tamaño fijo, lo que permite la adición o eliminación dinámica de elementos. Adicionalmente, estas listas ofrecen una variedad de métodos que facilitan la manipulación de sus elementos. Entre las operaciones disponibles se encuentran la adición, eliminación, búsqueda, ordenamiento, y otras funcionalidades que optimizan la gestión de la información contenida en la lista. [5]

1.5 UML

El Lenguaje de Modelado Unificado (UML) se erige como un estándar esencial para la representación visual de objetos, estados y procesos en el seno de un sistema. Desde una perspectiva, este lenguaje de modelado cumple la función de constituir un modelo para un proyecto, asegurando así una arquitectura de información sólidamente estructurada. Desde otro ángulo, UML facilita a los desarrolladores la tarea de presentar la descripción del sistema de una manera que resulte comprensible para aquellos que se encuentran fuera del ámbito especializado. En esencia, UML se emplea primordialmente en el desarrollo de software orientado a objetos. [6]

2.2 Clases

2.2.1 Clase Vehiculo

Se comenzó definiendo la clase “Vehiculo” a la cual se le otorgaron los atributos correspondientes. (Figura 2.2.1) Estos atributos representan información esencial sobre un vehículo, como su marca, modelo, matrícula, el año que se fabricó, su kilometraje, su color y precio de venta como también su precio de alquiler por día, si se encuentra disponible, sus fotos y un CampoEspecial para desarrollar con su herencia luego.

```
24 referencias
public class Vehiculo
{
    4 referencias
    public string matricula { get; set; }
    4 referencias
    public string modelo { get; set; }
    4 referencias
    public string marca { get; set; }
    3 referencias
    public string año { get; set; }
    3 referencias
    public int kilometros { get; set; }
    3 referencias
    public string color { get; set; }
    3 referencias
    public double precioVenta { get; set; }
    3 referencias
    public double precioAlquilerDia { get; set; }
    6 referencias
    public bool Activo { get; set; }
    3 referencias
    public string Imagen1 { get; set; }
    3 referencias
    public string Imagen2 { get; set; }
    3 referencias
    public string Imagen3 { get; set; }
    3 referencias
    public string CampoEspecial { get; set; }
}
```

Figura 2.2.1

También se desarrolla el método **DatosAMostrar** el cual retorna su matrícula, marca y modelo correspondiente. (Figura 2.2.2)

```
0 referencias
public string DatosMostrar
{
    get
    {
        return "Matricula: " + matricula + "Marca" + marca + "Modelo:" + modelo;
    }
}
```

Figura 2.2.2

2.2.2 Clases Hijas

Se crean las clases “Auto”, “Moto” y “Camion”, las cuales son hijas de la clase base “Vehiculo”, heredando así todos sus atributos y métodos. A su vez, a cada clase se le coloca un atributo diferente, a la clase “Auto” el atributo **pasajeros**, el cual devuelve la cantidad de pasajeros que pueden viajar en el vehículo (Figura 2.2.3). A la clase “Moto” se le atribuye las **cilindradas** (Figura 2.2.4), y por último a la clase “Camion” el atributo **carga**, que refiere a la capacidad de carga del vehículo. (Figura 2.2.5)

```
3 referencias
public class Auto : Vehiculo
{
    3 referencias
    public int pasajeros { get; set; }
}
```

Figura 2.2.3

```
3 referencias
public class Moto : Vehiculo
{
    3 referencias
    public int cilindradas { get; set; }
}
```

Figura 2.2.4

```
3 referencias
public class Camion : Vehiculo
{
    3 referencias
    public double carga { get; set; }
}
```

Figura 2.2.5

2.2.3 Clase Usuario

Se crea la clase “Usuarios”, con sus respectivos atributos como nombre de usuario, contraseña, permisos para ver la pestaña “Clientes”, “Usuarios”, “Ventas”, “Vehiculos” y “Alquileres”, como indica la figura 2.2.6.

```
12 referencias
public class Usuario
{
    8 referencias
    public string NombreUsuario { get; set; }
    4 referencias
    public string Contraseña { get; set; }
    2 referencias
    public bool VerClientes { get; set; }
    2 referencias
    public bool VerUsuarios { get; set; }
    2 referencias
    public bool VerVentas { get; set; }
    2 referencias
    public bool VerVehiculos { get; set; }
    2 referencias
    public bool VerAlquileres { get; set; }
}
```

Figura 2.2.6

2.2.4 Clase Cliente

La clase “Cliente” contiene atributos para ingresar a un cliente, como su documento nombre, apellido y dirección, como se indica la figura 2.2.7.

```
11 referencias
public class Cliente
{
    6 referencias
    public string Cedula { get; set; }
    6 referencias
    public string Nombre { get; set; }
    6 referencias
    public string Apellido { get; set; }
    5 referencias
    public string Direccion { get; set; }
}
```

Figura 2.2.7

En la clase Cliente, se encuentra dos funciones, que en conjunto validan si la cédula ingresada es Uruguaya. Figura 2.2.8

```
private static int validation_digit(string ci)
{
    var a = 0;
    var i = 0;
    if (ci.Length <= 6)
    {
        for (i = ci.Length; i < 7; i++)
        {
            ci = '0' + ci;
        }
    }
    for (i = 0; i < 7; i++)
    {
        a += (Int32.Parse("2987634"[i].ToString()) * Int32.Parse(ci[i].ToString())) % 10;
    }
    if (a % 10 == 0)
    {
        return 0;
    }
    else
    {
        return 10 - a % 10;
    }
}

1 referencia
public static bool Validate(string ci)
{
    var dig = ci[ci.Length - 1];
    ci = ci.Substring(0, ci.Length - 1);

    int validDigitCalculated = validation_digit(ci);
    return (Int32.Parse(dig.ToString()) == validDigitCalculated);
}
```

Figura 2.2.8

2.2.5 Clase Venta

En la clase “Venta”, se coloca los atributos de cedula del cliente, el nombre del usuario, la matricula del vehículo, la fecha de la venta y su precio. Algunos atributos que se colocarán de las clases de “Vehiculo”, “Cliente” y “Usuario”. (Figura 2.2.9)

```

4 referencias
public class Venta
{
    2 referencias
    public string Cedula { get; set; }
    2 referencias
    public string Matricula { get; set; }

    2 referencias
    public string NombreUsuario { get; set; }

    2 referencias
    public DateTime FechaVenta { get; set; }
    2 referencias
    public double Precio { get; set; }
}

```

Figura 2.2.9

2.2.6 Clase Alquiler

En la clase “Alquiler”, sucede algo similar a la clase “Venta”, se le atribuye cedula del cliente, matricula del vehículo, nombre de usuario, fecha de alquiler, la cantidad de días y su precio por día. Por último, un atributo para saber si fue devuelto, o no. Figura 2.2.10

```

4 referencias
public class Alquiler
{
    2 referencias
    public string Cedula { get; set; }
    3 referencias
    public string Matricula { get; set; }
    2 referencias
    public string NombreUsuario { get; set; }
    3 referencias
    public DateTime FechaAlquiler { get; set; }
    3 referencias
    public int Dias { get; set; }
    2 referencias
    public int Precio { get; set; }
    3 referencias
    public bool AutoDevuelto { get; set; }
}

```

Figura 2.2.10

La clase “Alquiler” también cuenta con un método llamado Estado, el cual se encarga de validar en el estado que se encuentra el mismo, si Atrasado, Vehículo devuelto o Al Día. Figura 2.2.11

```

public string Estado
{
    get
    {
        if (!AutoDevuelto && DateTime.Now > FechaAlquiler.AddDays(Dias))
        {
            return "Atrasado";
        }
        else if (!AutoDevuelto)
        {
            return "Al día";
        }
        else
        {
            return "Vehículo devuelto";
        }
    }
}

```

Figura 2.2.11

2.2.7 Clase BaseDeDatos.

Por último, se creó una clase base de datos, que simula ser una base de datos, pero con listas, donde se colocarán las listas de Alquileres, Ventas, Clientes, Usuarios y Vehículos (Figura 2.2.12). Se agregó un usuario admin, que tiene permiso para ver todas las pestañas y control total del sitio. Se agregan usuarios de ejemplo y vehículos también (Figura 2.2.13). Se crean métodos para listar los vehículos activos y otro método para guardar un usuario logeado (Figura 2.2.14).

```

54 referencias
public abstract class BaseDeDatos
{
    public static List<Vehiculo> listaVehiculos = new List<Vehiculo>();
    public static List<Cliente> listaClientes = new List<Cliente>();
    public static List<Usuario> listaUsuarios = new List<Usuario>();
    public static List<Venta> listaVentas = new List<Venta>();
    public static List<Alquiler> listaAlquileres = new List<Alquiler>();
    public static Usuario usuarioLogeado;
}

```

Figura 2.2.12

```

1 referencia
public static void CargarDatosIniciales()
{
    // Agregar un usuario administrador
    Usuario usuario = new Usuario();
    usuario.setNombreUsuario("Admin");
    usuario.setContrasena("Admin");
    usuario.setVerAlquileres(true);
    usuario.setVerVentas(true);
    usuario.setVerClientes(true);
    usuario.setVerUsuarios(true);
    usuario.setVerVehiculos(true);
    listaUsuarios.Add(usuario);

    // Agregar un usuario normal
    Usuario usuario1 = new Usuario();
    usuario1.setNombreUsuario("Vendedor");
    usuario1.setContrasena("Vendedor");
    usuario1.setVerAlquileres(true);
    usuario1.setVerVentas(true);
    usuario1.setVerClientes(true);
    usuario1.setVerUsuarios(false);
    usuario1.setVerVehiculos(true);
    listaUsuarios.Add(usuario1);
}

```

Figura 2.2.13

```

3 referencias
public static List<Vehiculo> ListadoVehiculosActivos()
{
    List<Vehiculo> vehiculosActivos = new List<Vehiculo>();
    foreach (var vehiculo in listaVehiculos)
    {
        if (vehiculo.Activo)
        {
            vehiculosActivos.Add(vehiculo);
        }
    }
    return vehiculosActivos;
}

1 referencia
public static void GuardarUsuarioLogeado(Usuario usuario)
{
    usuarioLogeado = usuario;
}

```

Figura 2.2.14

2.3 Site.Master

La página maestra, definida en el archivo "Site.master", está escrita en C# y tiene una clase asociada llamada "SiteMaster". La página define la estructura común para otras páginas del sitio. La sección <head> de la página contiene elementos como la codificación del documento, la configuración de la vista en dispositivos móviles, y un título que incluye la propiedad "Page.Title" de la página actual (Figura 2.3.1). Se incluyen referencias a varios scripts y estilos, gestionados por el control ScriptManager de ASP.NET, que facilita la administración de recursos cliente en la página. Además, se referencia un archivo de estilo CSS ubicado en la ruta "/Content/css". También se especifica un ícono para la pestaña del navegador mediante la etiqueta <link> que referencia un archivo de ícono ubicado en la ruta "/Iconshow-Transport-Sportscar-car-2.ico" (Figura 2.3.1).

```
Master Language="C#" AutoEventWireup="true" CodeBehind="Site.master.cs" Inherits="Obligatorio2023Prog2.SiteMaster"

<!DOCTYPE html>

<html lang="en">
<head runat="server">
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title><%: Page.Title %> - Obligatorio P II</title>

  <asp:Placeholder runat="server">
    <%: Scripts.Render("~/bundles/modernizr") %>
  </asp:Placeholder>

  <webresource runat="server" path="~/Content/css" />
  <link href="~/Iconshow-Transport-Sportscar-car-2.ico" rel="shortcut icon" type="image/x-icon" />

```

Figura 2.3.1

Dentro del cuerpo de la página, se utiliza un formulario ejecutándose en el servidor. Se utiliza el control ScriptManager para gestionar scripts, y se definen diversas referencias a scripts, tanto del framework como del sitio. La página incluye una barra de navegación (navbar) con enlaces a diferentes secciones del sitio, como "Clientes", "Vehiculos", "Ventas", "Alquileres" y "Usuarios" (Figura 2.3.2). El contenido específico de cada página se inserta en un espacio designado con el control ContentPlaceholder identificado como "MainContent". Finalmente, hay un pie de página con el año actual y el nombre de la aplicación, y se incluye el script de Bootstrap al final de la página. En resumen, esta página maestra proporciona la estructura común y la gestión de recursos compartidos para otras páginas en el sitio web, lo que facilita la consistencia y el mantenimiento del diseño.

```
<nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-dark bg-dark">
  <div class="container">
    <div class="collapse navbar-collapse d-sm-inline-flex justify-content-between">
      <ul class="navbar-nav flex-grow-1">
        <li class="nav-item"><a class="nav-link" id="lnkClientes" runat="server" href="~/Clientes">Clientes</a></li>
        <li class="nav-item"><a class="nav-link" id="lnkVehiculos" runat="server" href="~/Vehiculos">Vehiculos</a></li>
        <li class="nav-item"><a class="nav-link" id="lnkVentas" runat="server" href="~/Ventas">Ventas</a></li>
        <li class="nav-item"><a class="nav-link" id="lnkAlquileres" runat="server" href="~/Alquileres">Alquileres</a></li>
        <li class="nav-item"><a class="nav-link" id="lnkUsuarios" runat="server" href="~/Usuarios">Usuarios</a></li>
      </ul>
    </div>
  </div>
</nav>
```

Figura 2.3.2

2.4 Vehiculos

Agregando un nuevo elemento con un formulario web con página maestra, llamándolo Vehiculos, crea una primera “pestaña”, la cual a partir del archivo Vehiculos.aspx se puede agregar o modificar la interfaz visual de esa misma pestaña. Dentro de la pestaña Vehiculos.aspx, se puede crear o configurar diferentes atributos, como crear un div, y dentro colocarle una lista de botones de radio, los cuales te permiten seleccionar en este caso un tipo de vehículo en concreto, como, por ejemplo, “Moto”, “Auto” o “Camion”. Además, se le asigna un ID = “rblTipoVehiculo”, dato que luego le dará funcionalidad en la lógica y que, a su vez, debe ser único para poder referenciarlo. La propiedad runat=“server” indica que el control es un control de servidor y, por lo tanto, puede ser manipulado desde el código del servidor. AutoPostBack es la propiedad que al estar establecida en ‘true’, logra que la página realice un postback automáticamente al seleccionar un elemento de la lista de botones de radio. Un postback es una solicitud al servidor web para que vuelva a cargar la página. OnSelectedIndexChanged, especifica el nombre de un método en el código del servidor que se ejecutará cuando se produzca un cambio en la selección de la lista de botones de radio. En este caso, el método se llama rblTipoVehiculo_SelectedIndexChanged. Los atributos de “Value” especifican que cada valor se asocia a dicho elemento y “Selected” se encuentra en “True” para indicar que ese elemento esté seleccionado por defecto al cargar la página. (Figura 2.4.1)

```
<div class="row">
  <div class="col-lg-5">
    <asp:RadioButtonList ID="rblTipoVehiculo" runat="server" AutoPostBack="true" OnSelectedIndexChanged="rblTipoVehiculo_SelectedIndexChanged">
      <asp:ListItem Value="Moto" Selected="True">Moto</asp:ListItem>
      <asp:ListItem Value="Auto">Auto</asp:ListItem>
      <asp:ListItem Value="Camion">Camion</asp:ListItem>
    </asp:RadioButtonList>
  </div>
</div>
```

Figura 2.4.1

Luego de crear diferentes divs, con sus respectivos TextBox, para Marca del Vehiculo, Año del Vehiculo, Precio venta del Vehiculo, etc., y a cada uno, agregándole su ID, se creó una GridView, la cual también tiene su ID único, y OnRowCancelingEdit, OnRowDeleting, OnRowEditing, OnRowUpdating, son eventos del GridView que están asociados a métodos en el código del servidor que se ejecutarán cuando el usuario cancele la edición de una fila, elimine una fila, edite una fila o actualice una fila, respectivamente. (Figura 2.4.2)

```
<div class="row">
  <div class="col-lg-8">
    <asp:GridView ID="gvVehiculos" runat="server" Width="80%" BorderWidth="2px" CellSpacing="5"
      OnRowCancelingEdit="gvVehiculos_RowCancelingEdit"
      OnRowDeleting="gvVehiculos_RowDeleting"
      OnRowEditing="gvVehiculos_RowEditing"
      OnRowUpdating="gvVehiculos_RowUpdating"
      AutoGenerateColumns="false"
      DataKeyNames="Matricula">
    </asp:GridView>
  </div>
</div>
```

Figura 2.4.2

Un ejemplo dentro del GridView de como se colocó datos sobre CampoEspecial, sus ID únicos para la plantilla ItemTemplate, que contiene el control que se utilizará para mostrar los datos en la celda cuando no esté en modo de edición. Y la plantilla de EditItemTemplate, que contiene el control que se utilizará para editar los datos cuando la fila esté en modo de edición. <asp:Label>, <asp:TextBox>, <asp:Image>: Estos son controles que se utilizan para mostrar y editar datos en las celdas. La propiedad Text o ImageUrl se vincula a los datos utilizando la función Bind.

<asp:CommandField>: Agrega una columna con botones de comando, en este caso, botones para editar (ShowEditButton="true") y eliminar (ShowDeleteButton="true") filas. (Figura 2.4.3)

```
<asp:TemplateField HeaderText="CampoEspecial">
    <ItemTemplate>
        <asp:Label ID="lblCampoEspecial" runat="server" Text="<%# Bind("CampoEspecial") %>" /></asp:Label>
    </ItemTemplate>
</asp:TemplateField>

<asp:TemplateField HeaderText="Imagen1">
    <ItemTemplate>
        <asp:Image ID="imgImagen1" runat="server" ImageUrl="<%# Bind("Imagen1") %>" Height="200" Width="200"></asp:Image>
    </ItemTemplate>
    <EditItemTemplate>
        <asp:TextBox ID="txtImagen1Grid" runat="server" Text="<%# Bind("Imagen1") %>" /></asp:TextBox>
    </EditItemTemplate>
</asp:TemplateField>

<asp:CommandField ButtonType="Link" ShowEditButton="true" ShowDeleteButton="true" />
```

Figura 2.4.3

Dentro de Vehiculos.aspx, se encuentra Vehiculos.aspx.cs que es donde se encuentra el código que le da lógica a Vehiculos.aspx. Dentro se encuentra Page_Load.

El código en el evento Page_Load se encarga de configurar la visibilidad de los enlaces en el MasterPage, utilizando el método FindControl, según los permisos del usuario logeado y de cargar la lista de vehículos en el GridView si es la primera vez que se carga la página. (Figura 2.4.4)

```
0 referencias
protected void Page_Load(object sender, EventArgs e)
{
    Master.FindControl("lnkClientes").Visible = BaseDeDatos.usuarioLogeado.getVerClientes();
    Master.FindControl("lnkVehiculos").Visible = BaseDeDatos.usuarioLogeado.getVerVehiculos();
    Master.FindControl("lnkVentas").Visible = BaseDeDatos.usuarioLogeado.getVerVentas();
    Master.FindControl("lnkAlquileres").Visible = BaseDeDatos.usuarioLogeado.getVerAlquileres();
    Master.FindControl("lnkUsuarios").Visible = BaseDeDatos.usuarioLogeado.getVerUsuarios();

    if (!IsPostBack)
    {
        // Verificar si hay vehículos en la lista
        if (BaseDeDatos.listaVehiculos.Count > 0)
        {
            // Asignar la lista de vehículos como origen de datos para el GridView
            gvVehiculos.DataSource = BaseDeDatos.listaVehiculos;
            gvVehiculos.DataBind();
        }
    }
}
```

Figura 2.4.4

Se crea un botón para guardar datos (Figura 2.4.5), creando así un vehículo (auto, moto o camión), con sus atributos correspondientes. Se validan los datos, y se ejecuta un bloque de código diferente, utilizando *if*'s en caso de ser moto, auto o camión. Aquí el ejemplo de una moto. (Figura 2.4.6)

```
0 referencias
protected void btnGuardar_Click(object sender, EventArgs e)
{
    // Verificar si los TextBox están vacíos
    if (string.IsNullOrEmpty(txtMatricula.Text) ||
        string.IsNullOrEmpty(txtMarca.Text) ||
        string.IsNullOrEmpty(txtModelo.Text) ||
        string.IsNullOrEmpty(txtAño.Text) ||
        string.IsNullOrEmpty(txtKilometros.Text) ||
        string.IsNullOrEmpty(txtColor.Text) ||
        string.IsNullOrEmpty(txtPrecioVenta.Text) ||
        string.IsNullOrEmpty(txtPrecioAlquiler.Text) ||
        string.IsNullOrEmpty(txtCilindradas.Text) ||
        string.IsNullOrEmpty(txtCantPasajeros.Text) ||
        string.IsNullOrEmpty(txtToneladas.Text) ||
        string.IsNullOrEmpty(txtImagen1.Text) ||
        string.IsNullOrEmpty(txtImagen2.Text) ||
        string.IsNullOrEmpty(txtImagen3.Text))
    {
        lblMensajeError.Text = "Todos los campos son obligatorios. Complete la información.";
        return;
    }
}
```

Figura 2.4.5

```
if (rblTipoVehiculo.SelectedItem.Value == "Moto")
{
    Moto moto = new Moto();
    moto.setMatricula(txtMatricula.Text);
    moto.setModelo(txtModelo.Text);
    moto.setMarca(txtMarca.Text);
    moto.setAño(txtAño.Text);
    moto.setColor(txtColor.Text);

    // Validar y convertir la entrada de txtPrecioVenta.Text
    if (int.TryParse(txtPrecioVenta.Text, out int precioVenta))
    {
        moto.setPrecioVenta(precioVenta);
    }
    else
    {
        // Manejar la entrada no válida, mostrar un mensaje de error, etc.
        Response.Write("<script>alert('Precio de venta no válido')</script>");
        return;
    }

    // Repetir el mismo proceso para txtPrecioAlquiler.Text
    if (int.TryParse(txtPrecioAlquiler.Text, out int precioAlquiler))
    {
        moto.setPrecioAlquilerDia(precioAlquiler);
    }
    else
    {
        Response.Write("<script>alert('Precio de alquiler no válido')</script>");
        return;
    }

    moto.setImagen1(txtImagen1.Text);
    moto.setImagen2(txtImagen2.Text);
    moto.setImagen3(txtImagen3.Text);

    // Validar y convertir la entrada de txtCilindradas.Text
    if (int.TryParse(txtCilindradas.Text, out int cilindradas))
    {
        moto.setCilindradas(cilindradas);
    }
    else
    {
        Response.Write("<script>alert('Cilindradas no válidas')</script>");
        return;
    }

    BaseDeDatos.listaVehiculos.Add(moto);
}
```

Figura 2.4.6

A continuación, se explicará algunos de los métodos que actúan en el GridView, por ejemplo, `gvVehiculos_RowDeleting`, que obtiene la matrícula del vehículo seleccionado, busca y elimina el vehículo de la lista de Vehiculos en la base de datos, y por último, sale del modo edición y actualiza el GridView.

```
0 referencias
protected void gvVehiculos_RowDeleting(object sender, GridViewDeleteEventArgs e)
{
    string matricula = gvVehiculos.DataKeys[e.RowIndex].Values[0].ToString();

    foreach (var vehiculo in BaseDeDatos.listaVehiculos)
    {
        if (vehiculo.getMatricula() == matricula)
        {
            BaseDeDatos.listaVehiculos.Remove(vehiculo);
            break;
        }
    }

    this.gvVehiculos.EditIndex = -1;
    this.gvVehiculos.DataSource = BaseDeDatos.listaVehiculos;
    this.gvVehiculos.DataBind();
}
```

Figura 2.4.7

El método `gvVehiculos_RowCancelingEdit` se activa cuando se cancela la edición de una fila en el GridView. Sale del modo edición y actualiza.

```
protected void gvVehiculos_RowCancelingEdit(object sender, GridViewCancelEditEventArgs e)
{
    gvVehiculos.EditIndex = -1;
    gvVehiculos.DataSource = BaseDeDatos.listaVehiculos;
    gvVehiculos.DataBind();
}
```

Figura 2.4.8

Para editar, el método `gvVehiculos_RowEditing` es el que se activa cuando se inicia la edición de una fila en el GridView. Entra en modo de edición para la fila seleccionada y actualiza el GridView.

```
protected void gvVehiculos_RowEditing(object sender, GridViewEditEventArgs e)
{
    gvVehiculos.EditIndex = e.NewEditIndex;
    gvVehiculos.DataSource = BaseDeDatos.listaVehiculos;
    gvVehiculos.DataBind();
}
```

Figura 2.4.9

Este método se activa cuando se actualiza una fila en el GridView. Obtiene los nuevos valores editados de los controles TextBox, encuentra el vehículo correspondiente en la lista y actualiza sus propiedades con los nuevos valores. Luego, sale del modo de edición y actualiza el GridView.

```
protected void gvVehiculos_RowUpdating(object sender, GridViewUpdateEventArgs e)
{
    GridViewRow filaSeleccionada = gvVehiculos.Rows[e.RowIndex];
    string matricula = gvVehiculos.DataKeys[e.RowIndex].Values[0].ToString();

    string marca = (filaSeleccionada.FindControl("txtMarcaGrid") as TextBox).Text;
    string modelo = (filaSeleccionada.FindControl("txtModeloGrid") as TextBox).Text;
    string kilometros = (filaSeleccionada.FindControl("txtKilometrosGrid") as TextBox).Text;
    string precioVenta = (filaSeleccionada.FindControl("txtPrecioVenta") as TextBox).Text;
    string precioAlquilerDia = (filaSeleccionada.FindControl("txtPrecioAlquilerDia") as TextBox).Text;
    string campoEspecial = (filaSeleccionada.FindControl("txtCampoEspecialGrid") as TextBox).Text;
    string imagen1 = (filaSeleccionada.FindControl("txtImagen1Grid") as TextBox).Text;
    string imagen2 = (filaSeleccionada.FindControl("txtImagen2Grid") as TextBox).Text;
    string imagen3 = (filaSeleccionada.FindControl("txtImagen3Grid") as TextBox).Text;

    Vehiculo vehiculoToUpdate = BaseDeDatos.listaVehiculos.Find(v => v.getMatricula() == matricula);

    if (vehiculoToUpdate != null)
    {
        vehiculoToUpdate.setMarca(marca);
        vehiculoToUpdate.setModelo(modelo);
        vehiculoToUpdate.setAño((filaSeleccionada.FindControl("txtAñoGrid") as TextBox).Text);
        vehiculoToUpdate.setColor((filaSeleccionada.FindControl("txtColorGrid") as TextBox).Text);
        vehiculoToUpdate.setKilometros(Convert.ToInt32("txtKilometrosGrid.Text"));
        vehiculoToUpdate.setPrecioVenta(Convert.ToInt32("txtPrecioVentaGrid.Text"));
        vehiculoToUpdate.setPrecioAlquilerDia(Convert.ToInt32("txtPrecioAlquilerDia.Text"));
        vehiculoToUpdate.setCampoEspecial(campoEspecial);
        vehiculoToUpdate.setImagen1(imagen1);
        vehiculoToUpdate.setImagen2(imagen2);
        vehiculoToUpdate.setImagen3(imagen3);

        this.gvVehiculos.EditIndex = -1;
        this.gvVehiculos.DataSource = BaseDeDatos.listaVehiculos;
        this.gvVehiculos.DataBind();
    }
}
```

Figura 2.4.10

Este método se activa cuando cambia la selección en el RadioButtonList llamado rblTipoVehiculo. Dependiendo del valor seleccionado, se muestra u oculta ciertos controles TextBox (txtCilindradas, txtCantPasajeros, txtToneladas).

```
protected void rblTipoVehiculo_SelectedIndexChanged(object sender, EventArgs e)
{
    if (rblTipoVehiculo.SelectedItem.Value == "Moto")
    {
        txtCilindradas.Visible = true;
        txtCantPasajeros.Visible = false;
        txtToneladas.Visible = false;
    }
    if (rblTipoVehiculo.SelectedItem.Value == "Auto")
    {
        txtCilindradas.Visible = false;
        txtCantPasajeros.Visible = true;
        txtToneladas.Visible = false;
    }
    if (rblTipoVehiculo.SelectedItem.Value == "Camion")
    {
        txtCilindradas.Visible = false;
        txtCantPasajeros.Visible = false;
        txtToneladas.Visible = true;
    }
}
```

Figura 2.4.11

2.5 Usuarios

Para Usuarios.aspx.cs, el código se comporta de manera bastante similar al de Vehiculos. Agregar a un usuario implica rellenar los campos solicitados, tras validaciones (que no se encuentre vacío).

Luego el método btnGuardarUsuario_Click, hace uso de un **foreach**, y dentro de él verifica si los ítems se encuentran seleccionados, y a partir de ahí le indica un valor a cada caso. Figura 2.5.1

```
0 referencias
protected void btnGuardarUsuario_Click(object sender, EventArgs e)
{
    try
    {
        // Crear un nuevo usuario y asignar los valores
        Usuario nuevoUsuario = new Usuario();
        nuevoUsuario.setNombreUsuario(txtNombreUsuario.Text);
        nuevoUsuario.setContrasena(txtContrasena.Text);

        // Obtener los valores seleccionados del CheckBoxList
        foreach (ListItem item in cblPermisos.Items)
        {
            if (item.Selected)
            {
                switch (item.Value)
                {
                    case "VerClientes":
                        nuevoUsuario.setVerClientes(true);
                        break;
                    case "VerUsuarios":
                        nuevoUsuario.setVerUsuarios(true);
                        break;
                    case "VerVentas":
                        nuevoUsuario.setVerVentas(true);
                        break;
                    case "VerVehiculos":
                        nuevoUsuario.setVerVehiculos(true);
                        break;
                    case "VerAlquileres":
                        nuevoUsuario.setVerAlquileres(true);
                        break;
                }
            }
        }
    }
}
```

Figura 2.5.1

Luego en caso de que exista algún error en el registro, ejecuta un mensaje de error. Figura 2.5.2

```
        cargarUsuarios(),
    }
    catch (Exception ex)
    {
        // Manejar la excepción
        Response.Write($"Error al guardar el usuario: {ex.Message}");
    }
}
```

Figura 2.5.2

Con los métodos similares en Vehiculos, como RowEditing y RowUpdating, es capaz de editar los permisos de cada usuario.

2.6 Clientes

Clientes, tiene métodos que funcionan de manera similar a los registros de usuarios y vehículos, aunque tiene la particularidad del método Validate, para validar la cédula uruguaya. Figura 2.6.1

```
if (!Cliente.Validate(nuevoCliente.getCedula()))
{
    // Mostrar un mensaje de error
    lblMensajeCliente.Text = "La cédula no es válida. Por favor, ingrese una cédula uruguaya válida.";
    lblMensajeCliente.ForeColor = System.Drawing.Color.Red;
    lblMensajeCliente.Visible = true; // Mostrar el mensaje de error
    return;
}
```

Figura 2.6.1

2.7 Ventas

En la página de Ventas, el método btnGuardar_Click funciona creando una venta, la cual toma valores de Cédula(cboClientes), Matrícula(cboVehiculos) y NombreUsuario (lo adquiere de la base de datos, el usuario que se encuentre logeado en el sistema), Luego verifica si la fecha es válida, y define una variable precio, la cual es un entero. Añade la venta a la lista de Ventas, y luego mediante un foreach busca en base de datos ese vehículo (por la matricula) y lo pone en Inactivo (Activo = False). Con un break sale del bucle. Luego vuelve a enlazar el GridView actualizando la base de datos, e indicando un mensaje de “Venta ingresada correctamente” en verde. Figura 2.7.1

```
protected void btnGuardar_Click(object sender, EventArgs e)
{
    Venta venta = new Venta();
    venta.setCedula(cboClientes.SelectedItem.Value);
    venta.setMatricula(cboVehiculos.SelectedItem.Value);
    venta.setNombreUsuario(BaseDeDatos.usuarioLogeado.NombreUsuario);

    // Verificar si la fecha es válida antes de intentar convertirla
    DateTime fechaVenta;
    if (DateTime.TryParse(txtFecha.Text, out fechaVenta))
    {
        venta.setFechaVenta(fechaVenta);

        int precio;
        if (int.TryParse(lblPrecio.Text, out precio))
        {
            venta.setPrecio(precio);
        }

        BaseDeDatos.listaVentas.Add(venta);

        foreach (var vehiculo in BaseDeDatos.listaVehiculos)
        {
            if (vehiculo.getMatricula() == cboVehiculos.SelectedItem.Value)
            {
                vehiculo.Activo = false;
                break;
            }
        }

        // Vuelve a enlazar el GridView después de agregar la venta
        cboVehiculos.DataSource = BaseDeDatos.ListadoVehiculosActivos();
        cboVehiculos.DataTextField = "Matricula";
        cboVehiculos.DataBind();

        gridVentas.DataSource = BaseDeDatos.listaVentas;
        gridVentas.DataBind();

        lblMensaje.Text = "Venta ingresada correctamente";
        lblMensaje.ForeColor = System.Drawing.Color.Green;
        lblMensaje.Visible = true;
    }
}
```

Figura 2.7.1

2.8 Alquileres

Alquileres funciona de manera similar a la página de Ventas, únicamente que a diferencia cuenta con el método `calcularPrecioAlquiler()`, que calcula el precio total en base del precio por día multiplicado por la cantidad de días alquilados. Figura 2.8.1

```
public int calcularPrecioAlquiler()
{
    if (!string.IsNullOrEmpty(txtFechaAlquiler.Text) && !string.IsNullOrEmpty(txtDias.Text) && !string.IsNullOrEmpty(lblPrecio.Text))
    {
        int dias = Convert.ToInt32(txtDias.Text);
        int precioPorDia = Convert.ToInt32(lblPrecio.Text);

        int precioTotal = dias * precioPorDia;

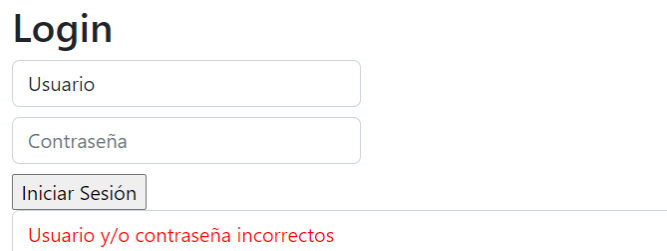
        return precioTotal;
    }

    return 0;
}
```

3. Resultados

El informe tiene varios periodos de prueba en las diferentes pestañas, tales como el Login, como primera página que aparece al entrar al proyecto, luego Vehiculos, como página principal donde aparece el catálogo de vehículos disponibles y que permite agregar, editar o eliminar motos, autos o camiones. Una página de Alquileres y otra de Ventas, donde se realizan las acciones tanto como de alquiler y de venta. Una pestaña Clientes, que te permite manipular diferentes acciones con ellos y también te genera un listado. Y, por último, la pestaña de Usuarios, únicamente modificable desde un login Admin, que te permite a su vez, manipular los permisos.

Login busca al usuario y en caso de no encontrarlo lanza el mensaje correspondiente. (Figura 3.1)



The screenshot shows a web form titled "Login". It contains two input fields: "Usuario" and "Contraseña". Below these fields is a button labeled "Iniciar Sesión". At the bottom of the form, there is a red error message that reads "Usuario y/o contraseña incorrectos".

Figura 3.1

Lo primero que se encuentra luego de iniciar sesión es una barra de navegación (Figura 3.2), para recorrer las diferentes páginas, y como página principal aparece Vehiculos. Su primera división dentro de la página, primero solicita una selección indicando si es moto, auto o camión (No puede ser de dos tipos a la vez), indica diferentes TextBox que funcionan para manipular los valores y asignarlos a sus respectivos datos, por ejemplo "Color de Vehiculo". Por último, un botón de guardar, el cual envía todos los datos recolectados creando en este caso una moto, con su Campo Especial que sería las cilindradas del Vehiculo.

Figura 3.2

En la pestaña Vehiculos, se encontrará una selección del tipo de vehículo, y campos para agregar datos que luego se tornarán en valores a cada atributo (Figura 3.3), con un botón de guardar que creará el registro de ese vehículo en concreto (Figura 3.4).

Catalogo de Vehiculos

☒ Moto

☐ Auto

☐ Camion

Matricula del vehiculo

Marca del Vehiculo

Modelo del Vehiculo

Año del Vehiculo

Figura 3.3

☐ Activo

Cilindradas del vehiculo

imagen 1 del Vehiculo

imagen 2 del Vehiculo

imagen 3 del Vehiculo

Guardar

Figura 3.4

Luego de cada registro, abajo aparecerá una lista con los vehículos registrados, sus atributos e imágenes. (Figura 3.5)

Vehículos registrados

Matricula	Marca	Modelo	Año	Kilometros	Color	PrecioVenta	PrecioAlquilerDia	Activo	CampoEspecial	Imagen1	Imagen2	Imagen3
ABC123	Chevrolet	Spark	2012	150000	Rojo	7900	500	True	Cant: Pasajeros 5			
										Editar	Eliminar	

Figura 3.5

Al hacer uso del método para editar, el mismo listado de vehículos entrará en modo de edición (Figura 3.6), lo que permite cambiar cualquier atributo, y al dar click en Actualizar, los datos serán actualizados con los datos nuevos. Luego saldrá del modo de edición y mostrará el listado, con el vehículo actualizado. (Figura 3.7)

Color	PrecioVenta	PrecioAlquilerDia	Activo	CampoEspecial	Imagen1	Imagen2	Imagen3	
<input type="text" value="Rojo"/>	<input type="text" value="7900"/>	<input type="text" value="500"/>	<input checked="" type="checkbox"/>	Cant: Pasajeros 5	<input type="text" value="https://s3.amazonaws.com"/>	<input type="text" value="https://s3.amazonaws.com"/>	<input type="text" value="https://s3.amazonaws.com"/>	Actualizar Cancelar

Figura 3.6

Vehículos registrados




Matricula	Marca	Modelo	Año	Kilometros	Color	PrecioVenta	PrecioAlquilerDia	Activo	CampoEspecial	Imagen1	Imagen2	Imagen3
ABC123	Chevrolet	Spark	2012	150000	Rojo	7900	1500	False	Cant: Pasajeros 5			
										Editar	Eliminar	

Figura 3.7

El registro de Clientes, funciona de igual forma que se registra un vehículo, la diferencia es que valida si la cédula es uruguaya, y en caso de no ser así, lo indica con un cartel de error en color rojo. Figura 3.8

Registro de Clientes

[Guardar Cliente](#)

Clientes Registrados

La cédula no es válida. Por favor, ingrese una cédula uruguaya válida.

Figura 3.8

Para Usuarios, el sistema actúa diferente, lo registras con un usuario y contraseña, y te permite seleccionar en que pestañas podrá interactuar ese usuario. Figura 3.9

Administracion de Usuarios

Vendedor5

.....

- ☒ Clientes
- ☐ Usuarios
- ☒ Ventas
- ☒ Vehiculos
- ☒ Alquileres

Guardar

Figura 3.9

Cuando se realiza el registro del usuario, aparece en una tabla con los usuarios ya registrados, indicando su usuario, contraseña y permisos. Figura 3.10

Vendedor5	vendedor5	True	False	True	True	True	Editar Eliminar
-----------	-----------	------	-------	------	------	------	---

Figura 3.10

Al hacer click en el botón Editar, te aparece los datos para editar y nuevamente seleccionas los permisos que podrá tener. Figura 3.11

Vende

Contrase

☒ ☐ ☒ ☒ ☒

[Actualizar](#) [Cancelar](#)

Figura 3.11

Y al finalizar de editar los datos, click en Actualizar y mostrará la tabla de datos actualizada. Figura 3.12

Vendedor5	vendedor5	True	False	True	False	False	Editar Eliminar
-----------	-----------	------	-------	------	-------	-------	---

Figura 3.12

Para realizar una Venta, el sistema te permite elegir que Cliente y que Vehiculo estará involucrado, registra una fecha, y el botón de Vender finaliza el registro de venta. Figura 3.13

Venta

Usuario:

Usuario: Admin

Clientes:

45866580- Nombre: Juan Perez

Vehiculos:

DEF456

Fecha:

dd/mm/aaaa



Vender

Figura 3.13

Luego se creará una tabla registrando cada venta, asociada a cierta cédula y cierta matrícula. Figura 3.14

Vehiculos vendidos registrados

Nombre Usuario	Cedula	Matricula	FechaVenta
Admin	45866580	DEF456	8/12/2023 16:48:06

Figura 3.14

Por último, la pestaña Alquileres, funciona de manera similar a la de Ventas, establece una conexión entre el Vehiculo y el Cliente, se selecciona una fecha y cantidad de días, y te informa del precio por día. Luego del botón Alquilar, te muestra el listado de Alquileres, detallando sus características, y mencionando su estado, si está Atrasado, Al día o el Vehiculo devuelto. Todas las características, se pueden editar de manera similar a los demás datos en las diferentes pestañas. Figura 3.15

Fecha:

08/12/2023



Días:

5

☐ Auto Devuelto

Alquilar

\$

500

Matricula	Cedula	FechaAlquiler	Dias	Precio	Devuelto	Estado	
MA125865	45809059	3/12/2023 16:50:26	3	10000	<input type="checkbox"/>	Atrasado	Editar Eliminar
SG321555	25688549	8/12/2023 16:50:26	3	12000	<input type="checkbox"/>	Al día	Editar Eliminar
RT9875578	45826584	26/11/2023 16:50:26	8	20000	<input checked="" type="checkbox"/>	Vehículo devuelto	Editar Eliminar
ABC123	45866580	8/12/2023 0:00:00	5	2500	<input type="checkbox"/>	Al día	Editar Eliminar

Figura 3.15

4. Conclusiones

Durante el desarrollo de este sistema de gestión para una automotora, se pudo aplicar de manera efectiva diferentes puntos para el logro del mismo. Se mencionan a continuación:

Complejidad y Funcionalidad Integral:

El proyecto aborda diversas funcionalidades, desde la gestión de vehículos hasta el seguimiento de alquileres y ventas. Esto indica una implementación integral que puede ser útil para una empresa de alquiler de vehículos.

Interfaz de Usuario Amigable:

Dado que hay varias páginas para diferentes aspectos de la aplicación (vehículos, login, clientes, alquileres, ventas), es esencial que la interfaz de usuario sea intuitiva y fácil de navegar. Una interfaz de usuario amigable contribuirá a una mejor experiencia del usuario.

Seguridad con Página de Login:

La página de login indica que se ha tenido en cuenta la seguridad, diferenciando entre usuarios y administradores. La implementación de roles y permisos adicionales para los administradores es crucial para garantizar la seguridad y la privacidad de la información.

Manejo de Imágenes y Atributos de Vehículos:

La capacidad de cargar imágenes y gestionar atributos como marca, modelo, color, etc., sugiere una atención al detalle en la representación de la información. La aplicación parece tener la capacidad de manejar diversos tipos de datos asociados a los vehículos.

Registro de Clientes y Detalles del Alquiler/Venta:

La página de clientes recopila información valiosa, y las páginas de alquiler y ventas solicitan datos específicos relacionados con estas transacciones. Un seguimiento detallado de la información de los clientes y las transacciones es esencial para una gestión eficiente.

Listado de Alquileres con Estado:

La página de listado de alquileres, clasificando entre en curso, finalizado o atrasado, brinda una visión clara del estado actual de las transacciones. Esto facilita la gestión y el seguimiento de los alquileres.

Posibles Mejoras:

Considera implementar funcionalidades adicionales como recordatorios automáticos para devoluciones atrasadas, generación de informes y estadísticas, y una interfaz administrativa más robusta.

Optimización de Rendimiento:

A medida que la base de datos crece, la optimización de consultas y la gestión eficiente de la base de datos son críticas para mantener un rendimiento óptimo de la aplicación.

5. Referencias

- [1] <https://aws.amazon.com/es/what-is/net/>
- [2] <https://learn.microsoft.com/es-es/aspnet/overview>
- [3] <https://learn.microsoft.com/es-es/visualstudio/get-started/visual-studio-ide?view=vs-2022>
- [4] <https://bsw.es/que-es-c/>
- [5] <https://oregoom.com/c-sharp/listas/>
- [6] <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/uml-lenguaje-unificado-de-modelado-orientado-a-objetos/>