

# Automotora Fast

Proyecto obligatorio 2023

Programación II



Prof. Gonzalo Duarte

Alumnos: Santiago Gutiérrez, Tobias Nuñez, Facundo Cardoso

2023



## Resumen

El proyecto desarrolla una aplicación web para la gestión de una empresa dedicada a la compra, venta y alquiler de vehículos (autos, motos y camiones). La implementación se llevó a cabo utilizando C#, ASP.Net framework, y las tecnologías web Bootstrap, HTML y CSS.

Al acceder a la página web, los usuarios se encuentran con la pestaña de Login, donde ingresan sus datos para acceder. Una vez ingresados en la página web, la primera sección disponible es la de "Vehículos". En esta sección, cada usuario puede registrar sus vehículos, diferenciándolos por tipo (autos, motos o camiones) y especificando sus características individuales. En la parte superior de la página, una barra de navegación facilita el acceso a diferentes secciones de la web, incluyendo "Vehículos", "Clientes", "Ventas", "Alquiler" y "Usuarios".

La sección de "Ventas" se dedica a la gestión de las ventas. Aquí, el usuario ingresa información clave como su cédula, la matrícula del vehículo y la fecha de publicación del auto para la venta.

En la sección "Alquiler", el proceso es similar, pero se incluye un campo adicional donde el usuario especifica la duración del alquiler en días.

Las clases "Cliente" y "Usuario". Clase "Cliente" incluye atributos esenciales como cédula, nombre, apellido y dirección. Por otro lado, la clase "Usuario" representa la información fundamental de los usuarios del sistema, con atributos como nombre de usuario, contraseña y permisos específicos para acceder a diversas partes del sistema.



# Indice

<b>1 Introducción:</b>	<b>1</b>
1.1 ¿Qué es un I.D.E?.....	1
1.2 ¿Qué es C#?.....	1
1.3 Atributos y Métodos en Programación.....	1
1.4 Bootstrap.....	2
1.5 Css.....	2
1.6 HTML.....	3
1.7 Herencia.....	3
1.8 Clases.....	3
<b>2 Desarrollo:</b>	<b>4</b>
2.1 Arquitectura y Tecnologías Utilizadas:.....	4
2.2 Clases del programa.....	5
Clase BaseDeDatos.....	5
Clase Cliente.....	5
Clase Vehiculo.....	6
Clase Alquiler.....	6
Clase Auto - Herencia de Vehiculo.....	7
Clase Camion - Herencia de Vehiculo.....	8
Clase Moto - Herencia de Vehiculo.....	8
Clase Usuario.....	9
Clase Venta.....	10
Clase Validaciones.....	10
2.3 Diagrama UML.....	12
<b>3 Funcionalidades:</b>	<b>13</b>
3.1 Página Usuarios.aspx y Usuarios.aspx.cs.....	13
Usuarios.aspx.....	13
Usuarios.aspx.cs.....	13
3.2 Página Ventas.aspx , Ventas.aspx.cs.....	17
Ventas.aspx.....	17
Ventas.aspx.cs.....	17
3.3 Página Alquileres.aspx y Alquileres.aspx.cs.....	21
Alquileres.aspx.....	21
Alquileres.aspx.cs.....	21
3.4 Página Login.aspx y Login.aspx.cs.....	27
Login.aspx.....	27
Login.aspx.cs.....	27
3.5 Interfaz de Usuario.....	29
<b>Resultados.....</b>	<b>33</b>
<b>Conclusión.....</b>	<b>34</b>
<b>Referencias.....</b>	<b>35</b>



# 1 Introducción:

## 1.1 ¿Qué es un I.D.E?

Un entorno de desarrollo integrado o IDE (Integrated Development Environment) es un espacio de trabajo virtual que se utiliza para el desarrollo y programación de aplicaciones de software. Gracias a las herramientas y mecanismos que aporta un IDE la tarea de programar es mucho más sencilla, ahorrando tiempo y consiguiendo que la productividad y eficiencia de los programadores y desarrolladores sea mucho más alta. Las características del IDE son diversas y suelen incluir la capacidad multiplataforma (para el desarrollo para distintos sistemas), el soporte para diferentes lenguajes de programación, la integración con un sistema para control de versiones, reconocimiento de sintaxis (para evitar errores al escribir código y acelerar el proceso de codificación), la integración con entornos de trabajo, capacidad de depuración de código, soporte para múltiples idiomas y posibilidad de importar y exportar proyectos.[1]

## 1.2 ¿Qué es C#?

C# Es un lenguaje de programación orientado a objetos, por lo tanto, los programas y aplicaciones realizados con este lenguaje están compuestos por clases, a partir de las cuales se crean objetos, que colaboran entre sí para resolver los problemas o necesidades de las aplicaciones. Dicho esto, en un programa con C#, incluso en el más básico, lo que nos vamos a encontrar son clases de programación orientada a objetos, en un número que dependerá de la complejidad de la aplicación. Unas clases más generales dependerán de otras más específicas mediante cualquier tipo de relación, como asociación, uso, composición y por supuesto herencia. [2]

## 1.3 Atributos y Métodos en Programación

Los atributos son variables que almacenan información sobre un objeto, mientras que los métodos son funciones que realizan tareas específicas en un programa. Ambos elementos son cruciales para definir el comportamiento y las propiedades de los objetos en un programa en C#. La correcta utilización de atributos y métodos permite crear programas más organizados y funcionales. [3]



## 1.4 Bootstrap

Bootstrap es un framework CSS utilizado en aplicaciones front-end — es decir, en la pantalla de interfaz con el usuario— para desarrollar aplicaciones que se adaptan a cualquier dispositivo. Tiene varios recursos para configurar los estilos de los elementos de la página de una manera simple y eficiente, además de facilitar la construcción de páginas que, al mismo tiempo, están adaptadas para la web y para dispositivos móviles. El propósito del framework es ofrecerle al usuario una experiencia más agradable cuando navega en un sitio. Bootstrap fue desarrollado por Twitter en 2010, para estandarizar las herramientas de la compañía. Inicialmente, se llamó Twitter Blueprint y, un poco más tarde, en 2011, se transformó en código abierto y su nombre cambió para Bootstrap. Desde entonces fue actualizado varias veces y ya se encuentra en la versión 4.4. El framework combina CSS y JavaScript para estilizar los elementos de una página HTML. Permite mucho más que, simplemente, cambiar el color de los botones y los enlaces. Esta es una herramienta que proporciona interactividad en la página, por lo que ofrece una serie de componentes que facilitan la comunicación con el usuario, como menús de navegación, controles de página, barras de progreso y más. Además de todas las características que ofrece el framework, su principal objetivo es permitir la construcción de sitios web responsive para dispositivos móviles. Esto significa que las páginas están diseñadas para funcionar en desktop, tablets y smartphones, de una manera muy simple y organizada.[4]

## 1.5 Css

CSS son las siglas en inglés para «hojas de estilo en cascada» (Cascading Style Sheets). Básicamente, es un lenguaje que maneja el diseño y presentación de las páginas web, es decir, cómo lucen cuando un usuario las visita. Funciona junto con el lenguaje HTML que se encarga del contenido básico de los sitios. Se les denomina hojas de estilo «en cascada» porque puedes tener varias y una de ellas con las propiedades heredadas (o «en cascada») de otras. Para muchas personas, una simple plantilla de blog es suficiente. Aun así, cuando quieras personalizar la apariencia de un sitio, necesitarás implementar CSS que, en conjunto con un buen CMS, te ayudará a potenciar el alcance de tu contenido. Con CSS, puedes crear reglas para decirle a tu sitio web cómo quieres mostrar la información y guardar los comandos para elementos de estilo (como fuentes, colores, tamaños, etc.) separados de los que configuran el contenido. Además, puedes crear formatos específicos útiles para comunicar tus ideas y producir experiencias más agradables, en el aspecto visual, para los usuarios del sitio web.[5]



## 1.6 HTML

HTML no es un lenguaje de programación; es un lenguaje de marcado que define la estructura de tu contenido. HTML consiste en una serie de elementos que usarás para encerrar diferentes partes del contenido para que se vean o comporten de una determinada manera. Las etiquetas de encierre pueden hacer de una palabra o una imagen un hipervínculo a otro sitio, se pueden cambiar palabras a cursiva, agrandar o achicar la letra, etc. [6]

## 1.7 Herencia

La herencia permite que se puedan definir nuevas clases basadas de unas ya existentes a fin de reutilizar el código, generando así una jerarquía de clases dentro de una aplicación. Si una clase deriva de otra, esta hereda sus atributos y métodos y puede añadir nuevos atributos, métodos o redefinir los heredados. La sintaxis de la herencia en C# es bastante sencilla. Para crear una clase que herede de otra, utilizamos el operador dos puntos (:) seguido del nombre de la clase de la que queremos heredar. [7]

## 1.8 Clases

Siempre que hablamos de clases, nos referimos a la programación orientada a objetos. Si lo queremos definir en una sola frase, una clase es un diseño que se puede utilizar para crear varios objetos individuales. Una clase define un grupo o conjunto de datos, en objetos se llaman atributos, que definen los objetos, así como un conjunto de comportamientos, las funciones o métodos del objeto, que lo manipulan y relacionan los objetos unos de otros. Los atributos y métodos son los miembros de la clase. Recordemos que para la gestión de clases debemos entender tres funciones claves para la programación orientada a objetos, la abstracción, la herencia y el polimorfismo. [8]

## 2 Desarrollo

El código desarrollado en C# con Visual Studio Community ofrece una aplicación web para gestionar una flota de vehículos de alquiler. Su objetivo principal es facilitar la administración de vehículos, permitiendo a los usuarios registrar información detallada, ver la lista de vehículos disponibles y agregar nuevos vehículos. Al iniciar sesión, los usuarios acceden a la sección principal, "Vehículos", donde pueden agregar información sobre autos, motos y camiones. La interfaz es fácil de usar, con una barra de navegación que permite acceder a diferentes secciones como "Clientes", "Ventas", "Alquiler" y "Usuarios". En las secciones "Ventas" y "Alquiler", los usuarios pueden gestionar transacciones, ingresando datos como la cédula del cliente, la matrícula del vehículo y la duración del alquiler. Las clases "Cliente" y "Usuario" proporcionan estructuras para manejar la información de clientes y usuarios del sistema. El código C# y Visual Studio ofrece una solución práctica para la gestión de flotas de vehículos de alquiler, brindando a los usuarios una herramienta fácil de usar para administrar cada aspecto del proceso, desde agregar nuevos vehículos hasta realizar ventas y alquileres.

### 2.1 Arquitectura y Tecnologías Utilizadas:

El sistema se ha desarrollado utilizando tecnologías web estándar como ASP.NET y C#. Se ha seguido el patrón Modelo-Vista-Controlador (MVC) para una estructura organizada.

ASP.NET y C#: Utilizados para el backend y la lógica del servidor.

- Bootstrap: Framework frontend que agiliza el diseño responsive de la interfaz.
- HTML y CSS: Definen la estructura y estilo de las páginas web.
- Esta combinación ofrece una base robusta para el desarrollo web, asegurando una arquitectura organizada y una interfaz moderna y atractiva.

#### Patrón Modelo-Vista-Controlador (MVC):

- **Modelo:** Representa la capa de datos y lógica de negocio. Incluye clases y estructuras que gestionan y manipulan los datos.
- **Vista:** Se compone de las páginas ASPX que definen la interfaz de usuario utilizando HTML y Bootstrap. La presentación y estilo están controlados por HTML y CSS.



- **Controlador:** Implementado en el código detrás de las páginas ASPX (archivos ASPX.CS), maneja la lógica de control, recibe la entrada del usuario y coordina las interacciones entre la vista y el modelo.

#### Ventajas del Patrón MVC:

1. **Separación de Responsabilidades:** Facilita la gestión y mantenimiento del código al dividir la aplicación en componentes con responsabilidades específicas.
2. **Reutilización de Código:** Permite la reutilización de componentes en diferentes partes de la aplicación, mejorando la eficiencia del desarrollo.
3. **Escalabilidad:** Proporciona una estructura organizada que facilita la incorporación de nuevas funcionalidades sin afectar otras partes del sistema.

## 2.2 Clases del programa

### Clase BaseDeDatos

Esta clase abstracta gestiona los datos de la web y cuenta con seis listas como atributos. La lista de Vehículos es estática y contiene objetos de la clase Vehículos. Lo mismo aplica para las listas de Usuarios, Clientes, Ventas y Alquileres. Además, posee el atributo usuarioLogeado, un objeto estático de la clase Usuario que representa al usuario autenticado. La clase cuenta con cinco métodos.

### Clase Cliente

La clase **Cliente** representa un cliente en el sistema y contiene información relacionada con su identificación, nombre, apellido, dirección y número de teléfono. A continuación, se presenta un análisis detallado:

- **Propiedades:**
  - **Cedula:** Identificación única del cliente.
  - **Nombre:** Nombre del cliente.
  - **Apellido:** Apellido del cliente.
  - **Direccion:** Dirección del cliente.
  - **Telefono:** Número de teléfono del cliente.
- **Constructor:**
  - Constructor predeterminado.
  - Constructor que acepta todos los parámetros necesarios para inicializar las propiedades del cliente.
- **Métodos de Acceso:**



- Métodos de acceso (**get** y **set**) para cada una de las propiedades del cliente.

## Clase Vehiculo

La clase **Vehiculo** representa los vehículos disponibles para alquiler en tu aplicación web. Aquí está el análisis detallado de la implementación:

- **Propiedades:**
  - **Matricula:** Identificación única de la matrícula del vehículo.
  - **Marca:** Marca del vehículo.
  - **Modelo:** Modelo específico del vehículo.
  - **PrecioVenta:** Precio de venta del vehículo.
  - **PrecioAlquilerDia:** Precio de alquiler por día del vehículo.
  - **Activo:** Indica si el vehículo está disponible para alquiler.
  - **Imagen1, Imagen2, Imagen3:** URLs o rutas de las imágenes asociadas al vehículo.
  - **Tipo:** Nuevo atributo que indica el tipo de vehículo (Moto, Auto, Camión).
- **Constructores:**
  - Constructor predeterminado.
  - Constructor que acepta todos los parámetros necesarios para inicializar las propiedades.
- **Métodos:**
  - Métodos de acceso (**get** y **set**) para cada propiedad, siguiendo convenciones de estilo de programación.
  - Se observa el uso de expresiones lambda (**=>**) para simplificar la sintaxis de los métodos de acceso.

## Clase Alquiler

La clase **Alquiler** modela un contrato de alquiler en tu aplicación web. Aquí hay un análisis detallado de la implementación:

- **Propiedades:**
  - **Cedula:** Representa la cédula del cliente que realiza el alquiler.
  - **Matricula:** Matrícula del vehículo alquilado.
  - **NombreUsuario:** Nombre del usuario asociado al alquiler.
  - **FechaAlquiler:** Fecha en que se realiza el alquiler.
  - **Dias:** Número de días que dura el alquiler.
  - **Precio:** Precio total del alquiler.



- **AutoDevuelto:** Indica si el auto ha sido devuelto.
- **Constructores:**
  - Constructor predeterminado.
  - Constructor que acepta todos los parámetros necesarios para inicializar las propiedades.
- **Métodos:**
  - Métodos de acceso (**get** y **set**) para cada propiedad, siguiendo convenciones de estilo de programación.
  - Se observa el uso de expresiones lambda (**=>**) para simplificar la sintaxis de los métodos de acceso.

## Clase Auto - Herencia de Vehiculo

La clase **Auto** extiende la clase base **Vehiculo** y añade propiedades específicas para vehículos tipo automóvil.

A continuación, se presenta el análisis detallado:

- **Propiedades Adicionales:**
  - **NumeroDePasajeros:** Nueva propiedad que representa la capacidad de pasajeros del automóvil.
- **Constructores:**
  - Constructor predeterminado.
  - Constructor que acepta todos los parámetros necesarios, incluyendo aquellos de la clase base (**Vehiculo**) y la propiedad adicional específica de **Auto**.
- **Métodos Adicionales:**
  - Métodos de acceso (**get** y **set**) para la propiedad específica de **Auto** (**NumeroDePasajeros**).
- **Comentarios Adicionales:**
  - La clase **Auto** hereda todas las propiedades y métodos de la clase base **Vehiculo**.
  - Se proporciona un constructor que utiliza la funcionalidad del constructor de la clase base para inicializar las propiedades compartidas.
  - Se añade una nueva propiedad específica de **Auto** para representar la cantidad de pasajeros que puede transportar.
  - Los métodos de acceso adicionales permiten gestionar la nueva propiedad.

Esta estructura de clases refleja la jerarquía y herencia adecuada, proporcionando una forma organizada y eficiente de representar distintos tipos de vehículos en tu sistema de alquiler. La herencia permite la reutilización de propiedades y métodos



comunes, mientras que las propiedades específicas de cada tipo de vehículo se manejan en las clases derivadas.

## Clase Camion - Herencia de Vehiculo

La clase **Camion** extiende la clase base **Vehiculo** y añade propiedades específicas para vehículos tipo camión. Aquí se presenta un análisis detallado:

- **Propiedades Adicionales:**
  - **ToneladasDeCarga:** Nueva propiedad que representa la capacidad de carga en toneladas del camión.
- **Constructores:**
  - Constructor predeterminado.
  - Constructor que acepta todos los parámetros necesarios, incluyendo aquellos de la clase base (**Vehiculo**) y la propiedad adicional específica de **Camion**.
- **Métodos Adicionales:**
  - Métodos de acceso (**get** y **set**) para la propiedad específica de **Camion** (**ToneladasDeCarga**).
- **Comentarios Adicionales:**
  - La clase **Camion** hereda todas las propiedades y métodos de la clase base **Vehiculo**.
  - Se proporciona un constructor que utiliza la funcionalidad del constructor de la clase base para inicializar las propiedades compartidas.
  - Se añade una nueva propiedad específica de **Camion** para representar la capacidad de carga en toneladas.
  - Los métodos de acceso adicionales permiten gestionar la nueva propiedad.

Al igual que con la clase **Auto**, la estructura de herencia y la adición de propiedades específicas para cada tipo de vehículo permiten una representación flexible y eficiente de los diferentes vehículos en el sistema de alquiler.

## Clase Moto - Herencia de Vehiculo

La clase **Moto** extiende la clase base **Vehiculo** y añade propiedades específicas para vehículos tipo moto. Aquí se presenta un análisis detallado:

- **Propiedades Adicionales:**
  - **Cilindrada:** Nueva propiedad que representa la cilindrada del motor de la moto.
- **Constructores:**



- Constructor predeterminado.
- Constructor que acepta todos los parámetros necesarios, incluyendo aquellos de la clase base (**Vehiculo**) y la propiedad adicional específica de **Moto**.
- **Métodos Adicionales:**
  - Métodos de acceso (**get** y **set**) para la propiedad específica de **Moto** (**Cilindrada**).
- **Comentarios Adicionales:**
  - La clase **Moto** hereda todas las propiedades y métodos de la clase base **Vehiculo**.
  - Se proporciona un constructor que utiliza la funcionalidad del constructor de la clase base para inicializar las propiedades compartidas.
  - Se añade una nueva propiedad específica de **Moto** para representar la cilindrada del motor.
  - Los métodos de acceso adicionales permiten gestionar la nueva propiedad.

La utilización de herencia en estas clases permite una representación eficiente y estructurada de los diferentes tipos de vehículos, compartiendo las propiedades y métodos comunes mientras permite extensiones específicas para cada tipo de vehículo.

## Clase Usuario

La clase **Usuario** representa a un usuario del sistema y contiene información asociada a su identificación, nombre de usuario, contraseña y permisos específicos para visualizar distintas secciones de la aplicación. A continuación, se presenta un análisis detallado:

- **Propiedades:**
  - **NombreUsuario:** Nombre del usuario.
  - **ApellidoUsuario:** Apellido del usuario.
  - **Contrasena:** Contraseña del usuario.
  - **VerClientes:** Permiso para visualizar la sección de clientes.
  - **VerUsuarios:** Permiso para visualizar la sección de usuarios.
  - **VerVentas:** Permiso para visualizar la sección de ventas.
  - **VerVehiculos:** Permiso para visualizar la sección de vehículos.
  - **VerAlquileres:** Permiso para visualizar la sección de alquileres.
  - **IdUsuario:** Identificación única del usuario.
  - **User:** Información adicional del usuario.
- **Constructor:**
  - Constructor predeterminado.



- Constructor que acepta todos los parámetros necesarios para inicializar las propiedades del usuario.
- **Métodos de Acceso:**
  - Métodos de acceso (**get** y **set**) para cada una de las propiedades del usuario.

## Clase Venta

La clase **Venta** modela una transacción de venta en el sistema, registrando detalles como la cédula del cliente, la matrícula del vehículo, el nombre del usuario que realizó la venta, la fecha de la venta y el precio asociado a la transacción.

A continuación, se presenta un análisis detallado:

- **Propiedades:**
  - **Cedula:** Cédula del cliente asociado a la venta.
  - **Matricula:** Matrícula del vehículo vendido.
  - **NombreUsuario:** Nombre del usuario que realizó la venta.
  - **FechaVenta:** Fecha en la que se realizó la venta.
  - **Precio:** Precio de la venta.
- **Constructor:**
  - Constructor predeterminado.
  - Constructor que acepta todos los parámetros necesarios para inicializar las propiedades de la venta.
- **Métodos de Acceso:**
  - Métodos de acceso (**get** y **set**) para cada una de las propiedades de la venta.

## Clase Validaciones

La clase **Validaciones** proporciona métodos estáticos para realizar validaciones comunes en distintos tipos de datos, como cédulas, matrículas, fechas y precios. A continuación, se presenta un análisis detallado de la clase:

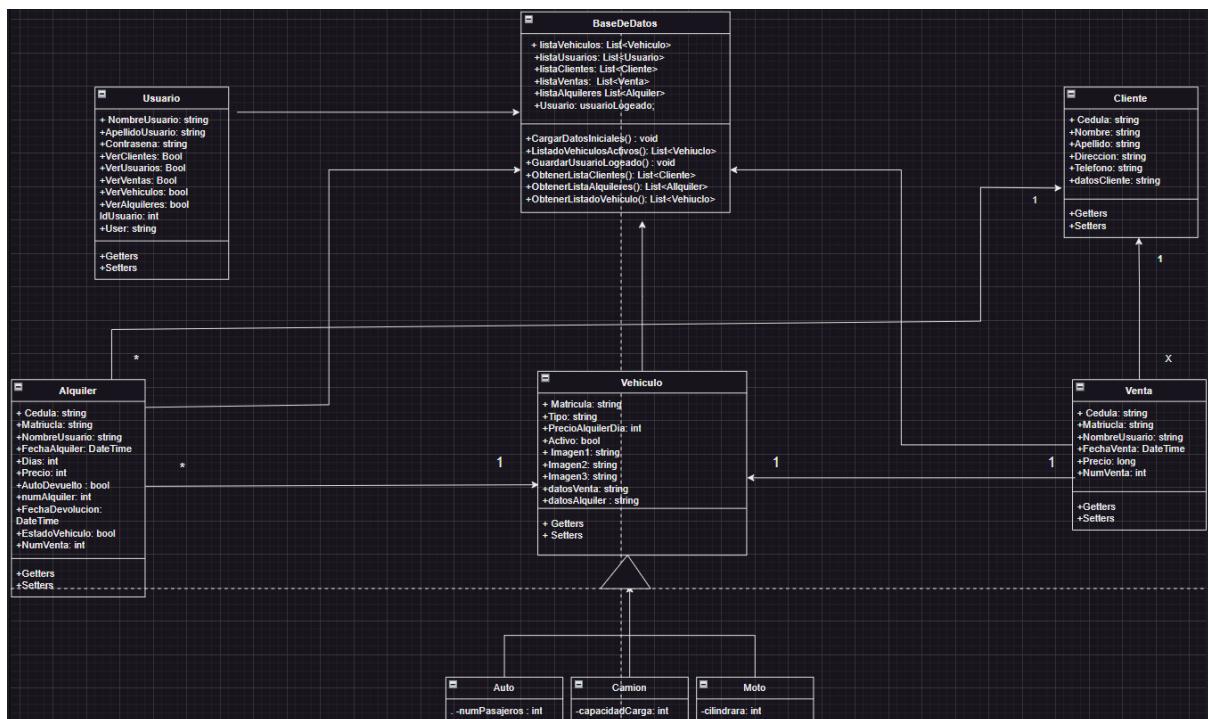
- **Método ValidarCedula:**
  - **Descripción:** Verifica si la cédula proporcionada es válida.
  - **Parámetros:**
    - **cedula** (string): Cédula a validar.
  - **Retorno: bool.** Devuelve **true** si la cédula es válida (no está vacía), de lo contrario, devuelve **false**.
- **Método ValidarMatricula:**
  - **Descripción:** Verifica si la matrícula proporcionada es válida.
  - **Parámetros:**



- **matricula** (string): Matrícula a validar.
- **Retorno: bool.** Devuelve **true** si la matrícula es válida (no está vacía), de lo contrario, devuelve **false**.
- **Método ValidarFecha:**
  - **Descripción:** Verifica si la cadena proporcionada puede convertirse en un objeto **DateTime**.
  - **Parámetros:**
    - **fecha** (string): Cadena que representa una fecha.
    - **fechaValidada** (out DateTime): Se asigna la fecha validada si la validación tiene éxito.
  - **Retorno: bool.** Devuelve **true** si la conversión es exitosa, de lo contrario, devuelve **false**.
- **Método ValidarPrecio:**
  - **Descripción:** Verifica si la cadena proporcionada puede convertirse en un entero.
  - **Parámetros:**
    - **precio** (string): Cadena que representa un precio.
    - **precioValidado** (out int): Se asigna el precio validado si la validación tiene éxito.
  - **Retorno: bool.** Devuelve **true** si la conversión es exitosa, de lo contrario, devuelve **false**.

## 2.3 Diagrama UML

Representa un sistema de gestión de alquiler y venta de vehículos con relaciones clave entre las clases. Las asociaciones entre Alquiler, Cliente, Usuario y Venta se visualizan, destacando la interconexión de registros de alquiler y venta con clientes, vehículos y usuarios. Además, se muestra la herencia entre las clases especializadas (Auto, Camion, Moto) y la clase general (Vehiculo). La clase BaseDeDatos actúa como un contenedor de listas. En resumen, el diagrama proporciona una visión clara de las relaciones y estructuras esenciales del sistema.



# 3 Funcionalidades

Funcionalidades Clave:

- Registro de Alquiler: Permite registrar nuevos alquileres asociados a clientes y vehículos seleccionados.
- Listado de Alquileres: Muestra una lista de alquileres con detalles como número, matrícula, tipo, marca, cliente, precio y estado del vehículo.

## 3.1 Página Usuarios.aspx y Usuarios.aspx.cs

Usuarios.aspx

La página **Usuarios.aspx** sirve como interfaz principal para la gestión de usuarios. Aquí, los usuarios del sistema pueden agregar nuevos usuarios, ver la lista existente y realizar acciones como editar o eliminar registros.

- **Diseño de la Página:**
  - Se ha diseñado una disposición limpia y estructurada utilizando elementos HTML y controles ASP.NET.
  - Contiene campos de entrada para información clave del usuario, como nombre, apellido y permisos.
  - Los controles de validación garantizan la integridad de los datos ingresados.

Usuarios.aspx.cs

El código detrás de la página **Usuarios.aspx** reside en **Usuarios.aspx.cs** y define la lógica asociada con la gestión de usuarios.

- **Page\_Load:**
  - Se encarga de configurar la visibilidad de los enlaces de navegación en función de los permisos del usuario actual.
  - Carga la lista de usuarios al cargar la página, pero solo si no es una devolución de formulario (**PostBack**).

```
protected void Page_Load(object sender, EventArgs e)
{
    Master.FindControl("lnkUsuarios").Visible = BaseDeDatos.usuarioLogeado.getVerUsuarios();
    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();

    if (!Page.IsPostBack)
    {
        this.gvUsuarios.DataSource = BaseDeDatos.listaUsuarios;
        this.gvUsuarios.DataBind();
    }
}
```



Figura 3.1 Page\_Load

- **btnGuardar\_Click:**
  - Maneja la lógica para agregar nuevos usuarios al sistema.
  - Realiza validaciones para asegurarse de que no se dupliquen nombres de usuario.
  - Actualiza la interfaz después de agregar un usuario.
- **gvUsuarios\_RowCancelingEdit,**
- **gvUsuarios\_RowDeleting,**
- **gvUsuarios\_RowUpdating,**
- **gvUsuarios\_RowEditing,**
- **gvUsuarios\_RowDataBound:**
  - Manejan eventos asociados con la edición, eliminación y actualización de registros en el **GridView** que muestra la lista de usuarios.

```
protected void btnGuardar_Click(object sender, EventArgs e)
{
    bool userCorrecto = true;

    foreach (Usuario usuario in BaseDeDatos.listaUsuarios)
    {
        if (usuario.User == txtUsername.Text)
        {
            lblError.Visible = true;
            txtUsername.Text = string.Empty;
            userCorrecto = false;
        }
    }

    if (userCorrecto == true)
    {
        Usuario usuario = new Usuario();

        int IdAnterior = BaseDeDatos.listaUsuarios.Any() ? BaseDeDatos.listaUsuarios.Max(usuario1 => usuario1.IdUsuario) : 0;
        int IdNuevo = IdAnterior + 1;
        usuario.setIdUsuario(IdNuevo);

        usuario.setUser(txtUsername.Text);
        usuario.setNombreUsuario(txtNombreUsuario.Text);
        usuario.setApellidoUsuario(txtApellidoUsuario.Text);
        usuario.setContrasena(txtPassword.Text);
        usuario.setVerClientes(chkVerClientes.Checked);
        usuario.setVer Usuarios(chkVerUsuarios.Checked);
        usuario.setVerVentas(chkVerVentas.Checked);
        usuario.setVerVehiculos(chkVerVehiculos.Checked);
        usuario.setVerAlquileres(chkVerAlquileres.Checked);

        BaseDeDatos.listaUsuarios.Add(usuario);
        this.gvUsuarios.DataSource = BaseDeDatos.listaUsuarios;
        this.gvUsuarios.DataBind();

        txtUsername.Text = string.Empty;
        txtNombreUsuario.Text = string.Empty;
        txtApellidoUsuario.Text = string.Empty;
        txtPassword.Text = string.Empty;
        chkVerClientes.Checked = false;
        chkVer Usuarios.Checked = false;
        chkVerVentas.Checked = false;
        chkVerVehiculos.Checked = false;
        chkVerAlquileres.Checked = false;
    }
}
```



*Figura 3.1 btnGuardar\_Click*

```
protected void gvUsuarios_RowCancelingEdit(object sender, GridViewCancelEventArgs e)
{
    this.gvUsuarios.EditIndex = -1;
    this.gvUsuarios.DataSource = BaseDeDatos.listaUsuarios;
    this.gvUsuarios.DataBind();
}
```

*Figura 3.1 gvUsuarios\_RowCancelingEdit*

```
0 referencias
protected void gvUsuarios_RowDeleting(object sender, GridViewDeleteEventArgs e)
{
    // Obtener el indice de la fila que se está eliminando
    int rowIndex = e.RowIndex;

    // Verificar si hay valores en las DataKeys y si la fila está dentro del rango
    if (gvUsuarios.DataKeys != null && rowIndex >= 0 && rowIndex < gvUsuarios.DataKeys.Count)
    {
        // Obtener el valor de "User" desde las DataKeys
        string username = gvUsuarios.DataKeys[rowIndex]? .Values["User"]?.ToString();

        if (username != null)
        {
            // Buscar y eliminar el usuario por su username
            Usuario usuarioAEliminar = BaseDeDatos.listaUsuarios.FirstOrDefault(u => u.getUser() == username);

            if (usuarioAEliminar != null)
            {
                BaseDeDatos.listaUsuarios.Remove(usuarioAEliminar);
                gvUsuarios.EditIndex = -1;
                gvUsuarios.DataSource = BaseDeDatos.listaUsuarios;
                gvUsuarios.DataBind();
            }
        }
    }
}
```

*Figura 3.1 gvUsuarios\_RowDeleting*

```
protected void gvUsuarios_RowDataBound(object sender, GridViewRowEventArgs e)
{
    if (gvUsuarios.EditIndex != -1 && e.Row.RowType == DataControlRowType.DataRow)
    {
        // Estás en modo de edición, deshabilita el botón de eliminación para este fila
        LinkButton lnkDelete = (LinkButton)e.Row.FindControl("lnkDelete");
        if (lnkDelete != null)
        {
            lnkDelete.Enabled = false;
            lnkDelete.CssClass = "disabled-link"; // Agrega una clase CSS para que sea visible
        }
    }
}
```



*Figura 3.1 gvUsuarios\_RowDataBound*

```
0 referencias
protected void gvUsuarios_RowUpdating(object sender, GridViewUpdateEventArgs e)
{
    GridViewRow filaSeleccionada = gvUsuarios.Rows[e.RowIndex];

    // Verificar si hay valores en las DataKeys
    if (gvUsuarios.DataKeys != null)
    {
        // Obtener el valor de "User" desde las DataKeys
        string username = gvUsuarios.DataKeys[e.RowIndex].Values["User"].ToString();

        if (username != null)
        {
            string nombre = (filaSeleccionada.FindControl("TextBox1") as TextBox).Text;
            string apellido = (filaSeleccionada.FindControl("txtApellidoGrid") as TextBox).Text;

            foreach (var usuario in BaseDeDatos.listaUsuarios)
            {
                if (usuario.getUser() == username)
                {
                    usuario.setNombreUsuario(nombre);
                    usuario.setApellidoUsuario(apellido);
                }
            }

            gvUsuarios.EditIndex = -1;
            gvUsuarios.DataSource = BaseDeDatos.listaUsuarios;
            gvUsuarios.DataBind();
        }
    }
}
```

*Figura 3.1 gvUsuarios\_RowUpdating*

```
0 referencias
protected void gvUsuarios_RowEditing(object sender, GridViewEditEventArgs e)
{
    this.gvUsuarios.EditIndex = e.NewEditIndex;
    this.gvUsuarios.DataSource = BaseDeDatos.listaUsuarios;
    this.gvUsuarios.DataBind();
}
```

*Figura 3.1 gvUsuarios\_RowEditing*



```

protected void gvUsuarios_RowDataBound(object sender, GridViewRowEventArgs e)
{
    if (gvUsuarios.EditIndex != -1 && e.Row.RowType == DataControlRowType.DataRow)
    {
        // Estás en modo de edición, deshabilita el botón de eliminación para este fila
        LinkButton lnkDelete = (LinkButton)e.Row.FindControl("lnkDelete");
        if (lnkDelete != null)
        {
            lnkDelete.Enabled = false;
            lnkDelete.CssClass = "disabled-link"; // Agrega una clase CSS para que sea visible pero no interactivo
        }
    }
}

```

*Figura 3.1 gvUsuarios\_RowDataBound*

## 3.2 Página Ventas.aspx , Ventas.aspx.cs

### Ventas.aspx

La página **Ventas.aspx** proporciona funcionalidades relacionadas con la venta de vehículos, incluida la selección de clientes y vehículos, y la visualización de precios.

- **Diseño de la Página:**
  - Interfaz simplificada con campos para seleccionar clientes, vehículos y fecha de venta.
  - Muestra el precio del vehículo seleccionado.

### Ventas.aspx.cs

- **Page\_Load:**
  - Carga las listas de clientes y vehículos al cargar la página.



```

protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        cboVehiculos.DataSource = BaseDeDatos.ListadoVehiculosActivos();
        cboVehiculos.DataTextField = "datosVenta";
        cboVehiculos.DataValueField = "Matricula";
        cboVehiculos.DataBind();

        cboClientes.DataSource = BaseDeDatos.listaClientes;
        cboClientes.DataTextField = "datosCliente";
        cboClientes.DataValueField = "Cedula";
        cboClientes.DataBind();
    }
}

0 referencias

```

*Figura 3.2 Page\_Load*

- **btnGuardar\_Click:**
  - Gestiona la lógica de registro de ventas, capturando la información seleccionada y actualizando la interfaz.

```

0 referencias
protected void btnGuardar_Click(object sender, EventArgs e)
{
    Venta venta = new Venta();
    venta.Cedula = cboClientes.SelectedItem.Value;
    venta.Matricula = cboVehiculos.SelectedItem.Value;
    venta.NombreUsuario = BaseDeDatos.usuarioLogeado.getUser();

    string Matricula = cboVehiculos.SelectedItem.Value;
    long precio = 0;
    foreach (var vehiculo in BaseDeDatos.listaVehiculos)
    {
        if (vehiculo.Matricula == Matricula)
        {
            precio = vehiculo.PrecioVenta;
            break;
        }
    }
    venta.Precio = precio;
    venta.FechaVenta = DateTime.Now;

    int numVentaAnterior = BaseDeDatos.listaVentas.Any() ? BaseDeDatos.listaVentas.Max(v => v.NumVenta) : 0;
    int numVentaNuevo = numVentaAnterior + 1;
    venta.NumVenta = numVentaNuevo;

    BaseDeDatos.listaVentas.Add(venta);

    foreach (var vehiculo in BaseDeDatos.listaVehiculos)
    {
        if (venta.getMatricula() == vehiculo.getMatricula())
        {
            vehiculo.Activo = false;
        }
    }
    this.gvVentas.DataSource = BaseDeDatos.listaVentas;
    this.gvVentas.DataBind();

    ActualizarListaVehiculos();
}
1 referencia

```



*Figura 3.2 btnGuardar\_Click*

- **cboVehiculos\_SelectedIndexChanged:**
  - Maneja el evento de cambio en la selección de vehículos, actualizando dinámicamente el precio mostrado.

```
protected void cboVehiculos_SelectedIndexChanged(object sender, EventArgs e)
{
    string Matricula = cboVehiculos.SelectedItem.Value;

    foreach (var vehiculo in BaseDeDatos.listaVehiculos)
    {
        if (vehiculo.Matricula == Matricula)
        {
            lblPrecio.Text = vehiculo.PrecioVenta.ToString();
            lblPrecio.Visible = true;
            lblPrecioSimbolo.Visible = true;
        }
    }
}
```

*Figura 3.2 CboVehiculos\_SelectedIndexChanged*

```
protected string ObtenerNombreApellidoCliente(object documentoObject)
{
    string documento = documentoObject.ToString();
    Cliente cliente = BaseDeDatos.ObtenerListaClientes().Find(v => v.Cedula == documento);

    return cliente != null ? cliente.Nombre.ToString() + " " + cliente.Apellido.ToString() : "No disponible";
}
```

*Figura 3.2 ObtenerNombreApellidoCliente*

```
protected string ObtenerTipoVehiculo(object matriculaObject)
{
    string matricula = matriculaObject.ToString();
    Vehiculo vehiculo = BaseDeDatos.ObtenerListaVehiculos().Find(v => v.Matricula == matricula);

    return vehiculo != null ? vehiculo.Tipo.ToString() : "No existe";
}
```

*Figura 3.2 ObtenerTipoVehiculo*



```

protected string ObtenerMarcaVehiculo(object matriculaObject)
{
    string matricula = matriculaObject.ToString();
    Vehiculo vehiculo = BaseDeDatos.ObtenerListaVehiculos().Find(v => v.Matricula == matricula);
    return vehiculo != null ? vehiculo.Marca : "No disponible";
}

```

*Figura 3.2 ObtenerMarcaVehiculo*

```

protected string ObtenerDocumentoCliente(object documentoObject)
{
    string documento = documentoObject.ToString();
    Cliente cliente = BaseDeDatos.ObtenerListaClientes().Find(v => v.Cedula == documento);

    return cliente != null ? cliente.Cedula.ToString() : "No disponible";
}

```

*Figura 3.2 ObtenerDocumentoCliente*

```

protected string ObtenerPrecioVentaVehiculo(object matriculaObject)
{
    string matricula = matriculaObject.ToString();
    Vehiculo vehiculo = BaseDeDatos.ObtenerListaVehiculos().Find(v => v.Matricula == matricula);

    return vehiculo != null ? "$" + vehiculo.PrecioVenta.ToString() : "No disponible";
}

```

*Figura 3.2 ObtenerPrecioVentaVehiculo*

```

private void ActualizarListaVehiculos() // //actualiza visualmente la lista de vehiculos a vender
{
    var vehiculosDisponibles = BaseDeDatos.ListadoVehiculosActivos();

    cboVehiculos.DataSource = vehiculosDisponibles;
    cboVehiculos.DataTextField = "datosVenta";
    cboVehiculos.DataValueField = "Matricula";
    cboVehiculos.DataBind();
}

```

*Figura 3.2 ActualizarListaVehiculo*



### 3.3 Página Alquileres.aspx y Alquileres.aspx.cs

#### Alquileres.aspx

La página Alquileres.aspx permite la gestión de alquileres de vehículos, incluyendo el registro de nuevos alquileres y la visualización de alquileres existentes.

#### Diseño de la Página:

- Diseño estructurado con campos para la selección de clientes y vehículos, ingreso de la fecha y duración del alquiler.
- Visualización del precio del alquiler según la duración y tarifas del vehículo seleccionado.
- Listado de alquileres con detalles como número, matrícula, tipo, marca, cliente, precio y estado del vehículo.

#### Alquileres.aspx.cs

#### Page\_Load:

- Carga los datos iniciales como la lista de clientes, la lista de vehículos activos y la lista de alquileres.
- Enlaza la lista de alquileres al control GridView para su visualización.

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        lstClientes.DataSource = BaseDeDatos.listaClientes;
        lstClientes.DataTextField = "datosCliente";
        lstClientes.DataValueField = "Cedula";
        lstClientes.DataBind();

        cboVehiculos.DataSource = BaseDeDatos.ListadoVehiculosActivos();
        cboVehiculos.DataTextField = "datosAlquiler";
        cboVehiculos.DataValueField = "Matricula";
        cboVehiculos.DataBind();

        CargarGridAlquileres();
    }
}
```

Figura 3.3 Page\_Load



### cboVehiculos\_SelectedIndexChanged:

- Maneja el evento de cambio en la selección de vehículos, actualizando dinámicamente el precio del alquiler mostrado.

```
protected void cboVehiculos_SelectedIndexChanged(object sender, EventArgs e)
{
    string Matricula = cboVehiculos.SelectedItem.Value;

    foreach (var vehiculo in BaseDeDatos.listaVehiculos)
    {
        if (vehiculo.Matricula == Matricula)
        {
            lblPrecio.Text = vehiculo.PrecioVenta.ToString();
            lblPrecio.Visible = true;
            lblPrecioSimbolo.Visible = true;
        }
    }
}
```

Figura 3.3 cboVehiculos\_SelectedIndexChanged

### btnGuardar\_Click:

- Gestiona la lógica para registrar nuevos alquileres.
- Calcula automáticamente el precio total del alquiler en función de la duración y la tarifa diaria del vehículo.
- Actualiza la interfaz después de agregar un nuevo alquiler.



```

0 referencias
protected void btnGuardar_Click(object sender, EventArgs e)
{
    Alquiler alquiler = new Alquiler();

    alquiler.setCedula(lstClientes.SelectedItem.Value);
    alquiler.setMatricula(cboVehiculos.SelectedItem.Value);
    alquiler.setNombreUsuario(BaseDeDatos.usuarioLogeado.getUser());

    string Matricula = cboVehiculos.SelectedItem.Value;

    int cantDias = Convert.ToInt32(txtDiasAlquiler.Text);
    alquiler.setDias(cantDias);

    int precioTotal = 0;
    foreach (var vehiculo in BaseDeDatos.listaVehiculos)
    {
        if (alquiler.getMatricula() == vehiculo.getMatricula())
        {
            precioTotal = vehiculo.getPrecioAlquilerDia() * cantDias;
        }
    }
    alquiler.Precio = precioTotal;

    DateTime fechaInicio = Convert.ToDateTime(txtFecha.Text);
    alquiler.FechaAlquiler = fechaInicio;

    int numAlquilerAnterior = BaseDeDatos.listaAlquileres.Any() ? BaseDeDatos.listaAlquileres.Max(alquiler1 => alquiler1.numAlquiler) : 0;
    int numAlquilerNuevo = numAlquilerAnterior + 1;
    alquiler.setNumAlquiler(numAlquilerNuevo);

    // Obtener la fecha de devolución esperada (por ejemplo, sumando los días de alquiler a la fecha de inicio)
    DateTime fechaDevolucionEsperada = fechaInicio.AddDays(cantDias);

    // Determinar el estado del vehículo
    bool estadoVehiculo = (DateTime.Now <= fechaDevolucionEsperada);

    alquiler.EstadoVehiculo = estadoVehiculo;

    BaseDeDatos.listaAlquileres.Add(alquiler);
}

```

```

foreach (var vehiculo in BaseDeDatos.listaVehiculos)
{
    if (alquiler.getMatricula() == vehiculo.getMatricula())
    {
        vehiculo.Activo = false;
    }
}

this.gridAlquileres.DataSource = BaseDeDatos.listaAlquileres;
this.gridAlquileres.DataBind();

ActualizarListaVehiculos();
LimpiarCampos();
}

```

*Figura 3.3 btnGuardar\_Click*

### ActualizarListaVehiculos:

- Actualiza visualmente la lista de vehículos disponibles después de realizar un alquiler.



```

    private void ActualizarListaVehiculos() // //actualista visualmente la lista de vehiculos a vender
{
    var vehiculosDisponibles = BaseDeDatos.ListadoVehiculosActivos();

    cboVehiculos.DataSource = vehiculosDisponibles;
    cboVehiculos.DataTextField = "Matricula";
    cboVehiculos.DataValueField = "Matricula";
    cboVehiculos.DataBind();
}

```

*Figura 3.3 ActualizarListaVehiculos*

#### **ActualizarPrecio:**

- Actualiza dinámicamente el precio del alquiler en la interfaz en función de la duración y la tarifa diaria del vehículo seleccionado.

```

    private void ActualizarPrecio()
{
    string Matricula = cboVehiculos.SelectedItem.Value;
    Alquiler alquiler = new Alquiler();

    int precioTotal = 0;
    int cantDias = 0;

    if (txtDiasAlquiler.Text != "")
    {
        cantDias = Convert.ToInt32(txtDiasAlquiler.Text);
    }
    alquiler.setDias(cantDias);

    foreach (var vehiculo in BaseDeDatos.listaVehiculos)
    {
        if (vehiculo.Matricula == Matricula)
        {
            precioTotal = vehiculo.getPrecioAlquilerDia() * cantDias;
            lblPrecio.Text = precioTotal.ToString();
            lblPrecio.Visible = true;
            lblPrecioSimbolo.Visible = true;
        }
    }
}

```

*Figura 3.3 ActualizarPrecio*

#### **CargarGridAlquileres:**

- Carga la lista de alquileres y la enlaza al control GridView para su visualización.



```

private void CargarGridAlquileres()
{
    // Obtener la lista de alquileres (deberías tener una lista real en tu aplicación)
    var listaAlquileres = BaseDeDatos.ObtenerListaAlquileres();

    // Vincular la lista al GridView
    gridAlquileres.DataSource = listaAlquileres;
    gridAlquileres.DataBind();
}

```

*Figura 3.3 CargarGridAlquileres*

- Restablece los campos de entrada y etiquetas de precio después de realizar un alquiler.

```

private void LimpiarCampos()
{
    lstClientes.SelectedIndex = -1;
    cboVehiculos.SelectedIndex = -1;
    txtFecha.Text = string.Empty;
    txtDiasAlquiler.Text = string.Empty;
    lblPrecio.Text = string.Empty;
    lblPrecioSimbolo.Visible = false;
    lblPrecio.Visible = false;
}

```

*Figura 3.3 LimpiarCampos*

```

protected void txtDiasAlquiler_TextChanged(object sender, EventArgs e)
{
    ActualizarPrecio();
}

```

*Figura 3.3 txtDiasAlquiler\_TextChanged*



```

protected string ObtenerPrecioAlquilerVehiculo(object matriculaObject, object numAlquilerObject)
{
    string matricula = matriculaObject.ToString();
    int numAlquiler = Convert.ToInt32(numAlquilerObject);

    // Buscar el vehículo y el alquiler
    Vehiculo vehiculo = BaseDeDatos.ObtenerListaVehiculos().Find(v => v.Matricula == matricula);
    Alquiler alquiler = BaseDeDatos.ObtenerListaAlquileres().Find(a => a.numAlquiler == numAlquiler);

    // Verificar si vehículo y alquiler son nulos
    if (vehiculo != null && alquiler != null)
    {
        int PrecioTotal = vehiculo.PrecioAlquilerDia * alquiler.Dias;
        return "$" + PrecioTotal.ToString();
    }
    else
    {
        return "No disponible";
    }
}

```

*Figura 3.3 ObtenerPrecioAlquilerVehiculo*

```

protected string ObtenerDocumentoCliente(object documentoObject)
{
    string documento = documentoObject.ToString();
    Cliente cliente = BaseDeDatos.ObtenerListaClientes().Find(v => v.Cedula == documento);

    return cliente != null ? cliente.Cedula.ToString() : "No disponible";
}

1 referencia

```

*Figura 3.3 ObtenerDocumentoCliente*

```

1 referencia
protected string ObtenerMarcaVehiculo(object matriculaObject)
{
    string matricula = matriculaObject.ToString();
    Vehiculo vehiculo = BaseDeDatos.ObtenerListaVehiculos().Find(v => v.Matricula == matricula);
    return vehiculo != null ? vehiculo.Marca : "No disponible";
}

```

*Figura 3.3 ObtenerMarcaVehiculo*

```

1 referencia
protected string ObtenerTipoVehiculo(object matriculaObject)
{
    string matricula = matriculaObject.ToString();
    Vehiculo vehiculo = BaseDeDatos.ObtenerListaVehiculos().Find(v => v.Matricula == matricula);

    return vehiculo != null ? vehiculo.Tipo.ToString() : "No existe";
}

```

*Figura 3.3 ObtenerTipoVehiculo*



```

    I Referencia
protected string ObtenerNombreApellidoCliente(object documentoObject)
{
    string documento = documentoObject.ToString();
    Cliente cliente = BaseDeDatos.ObtenerListaClientes().Find(v => v.Cedula == documento);

    return cliente != null ? cliente.Nombre.ToString() + " " + cliente.Apellido.ToString() : "No disponible";
}

```

*Figura 3.3 ObtenerNombreApellidoCliente*

## 3.4 Página Login.aspx y Login.aspx.cs

### Login.aspx

La página Login.aspx proporciona la funcionalidad de inicio de sesión para los usuarios del sistema.

#### Diseño de la Página:

- Interfaz con campos para ingresar el nombre de usuario y la contraseña.
- Botón de inicio de sesión que activa la lógica de autenticación.

### Login.aspx.cs

#### Page\_Load:

- Configura la visibilidad de los enlaces de navegación en función de los permisos del usuario actual.
- Carga datos iniciales si no es un PostBack.

```

    O Referencias
protected void Page_Load(object sender, EventArgs e)
{
    Master.FindControl("lnkClientes").Visible = false;
    Master.FindControl("lnkVehiculos").Visible = false;
    Master.FindControl("lnkVentas").Visible = false;
    Master.FindControl("lnkAlquileres").Visible = false;
    Master.FindControl("lnkUsuarios").Visible = false;

    if (!Page.IsPostBack)
        BaseDeDatos.CargarDatosIniciales();
}

```



### 3.4 Login.aspx.cs

#### btnGuardar\_Click:

- Maneja la lógica de inicio de sesión, autenticando a los usuarios y redirigiéndolos a las páginas correspondientes según sus permisos.
- Muestra un mensaje de error en caso de credenciales incorrectas.

```
0 referencias
protected void btnGuardar_Click(object sender, EventArgs e)
{
    bool loginCorrecto = false;
    foreach (var usuario in BaseDeDatos.listaUsuarios)
    {
        if (usuario.getNombreUsuario() == txtUsuario.Text && usuario.getContrasena() == txtContrasena.Text)
        {
            BaseDeDatos.GuardarUsuarioLogeado(usuario);
            if (BaseDeDatos.usuarioLogeado.getVerVehiculos())
                Response.Redirect("Vehiculos.aspx");

            if (BaseDeDatos.usuarioLogeado.getVer Usuarios())
                Response.Redirect("Usuarios.aspx");

            if (BaseDeDatos.usuarioLogeado.getVerClientes())
                Response.Redirect("Clientes.aspx");

            if (BaseDeDatos.usuarioLogeado.getVerVentas())
                Response.Redirect("Ventas.aspx");

            if (BaseDeDatos.usuarioLogeado.getVerAlquileres())
                Response.Redirect("Alquileres.aspx");

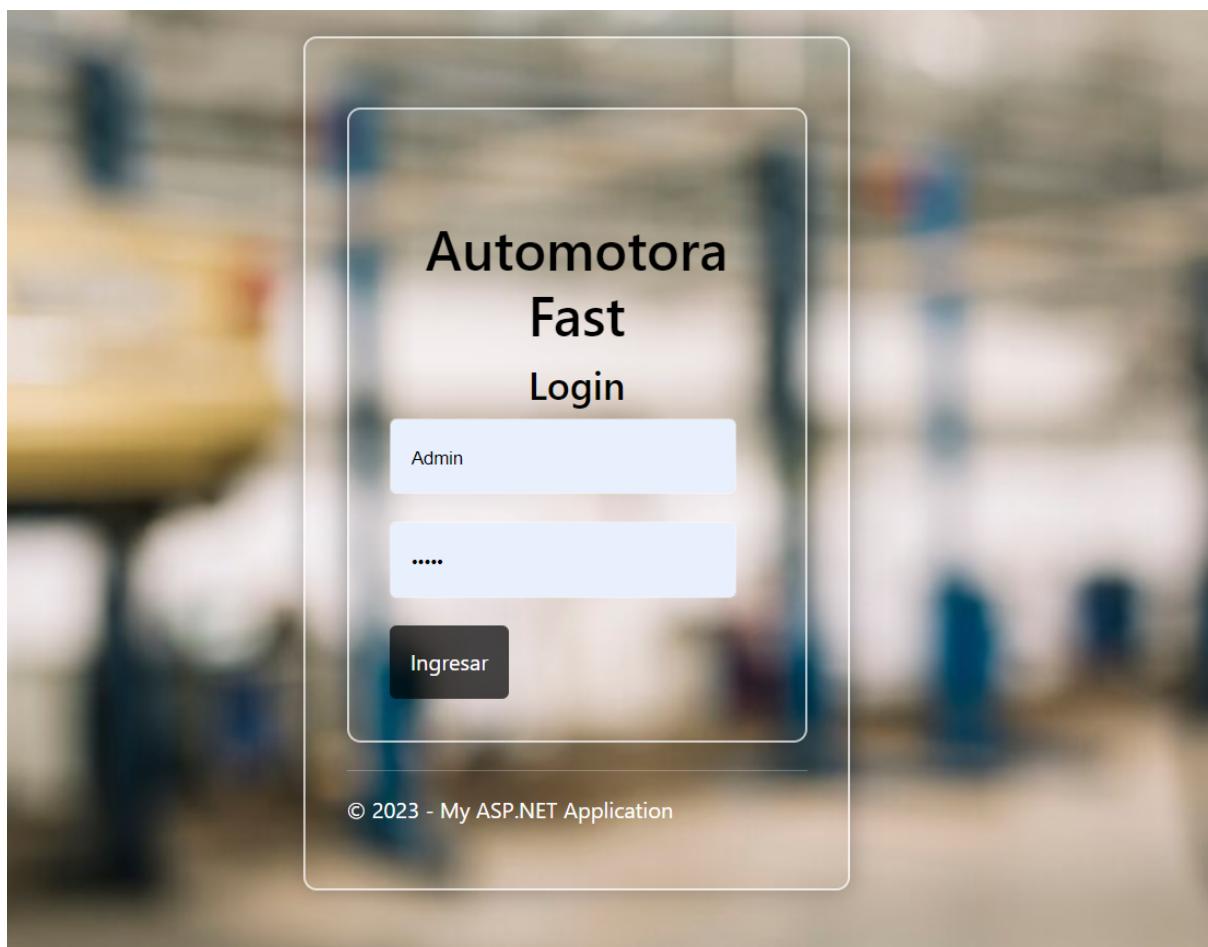
            loginCorrecto = true;
        }
    }
    if (!loginCorrecto)
    {
        lblError.Visible = true;
        Response.Write("<script>alert('usuario y/o contraseña incorrectos')</script>");
    }
}
```

#### 3.4 btnGuardar\_Click:



### 3.5 Interfaz de Usuario

Se ha diseñado una interfaz intuitiva con elementos visuales atractivos. La selección de clientes y vehículos se realiza mediante listas desplegables, y se proporcionan validaciones para asegurar la integridad de los datos. La interfaz presenta un diseño atractivo y transparente. Se han aplicado efectos visuales para mejorar la experiencia del usuario.



3.5 *Login*

Cuentas - Vehículos - Ventas - Alquileres - Usuarios

### Registro de Vehículos

Matrícula del Vehículo

Marca del Vehículo

Modelo del Vehículo

Precio de venta

Precio de alquiler diario

Imagen1 del Vehículo

Imagen2 del Vehículo

Imagen3 del Vehículo

Moto  
 Auto  
 Camión

Cilindrada

**Guardar**

**Listado de Vehículos**

**Guardar**

### Listado de Vehículos

Matrícula	Marca	Modelo	Precio Venta	Precio Alquiler	Activo	Imagen 1	
MA458855	Ferrari	F40	1000000	100	True		<a href="#">Editar</a> <a href="#">Cerrar</a>
TG945384	Chery	TiGO	95600	180	True		<a href="#">Editar</a> <a href="#">Eliminar</a>
FR46665	Fiat	UNO	857588	120	True		<a href="#">Editar</a> <a href="#">Eliminar</a>

© 2023 - My ASP.NET Application

### 3.5 Vehículos

Cuentas Vehículos Ventas Alquileres Usuarios

### Agregar un nuevo Cliente

Nombre del Cliente
Apellido del Cliente
Documento del Cliente
Dirección del Cliente
Teléfono del Cliente

### Lista de Clientes

Documento	Nombre	Apellido	Dirección	Teléfono	Editar	Eliminar
4586658-0	Tobias	Nufiez	dr Edye 3456	091641844	<a href="#">Editar</a>	<a href="#">Eliminar</a>
4589998-9	Santiago	Gutierrez	dr Edye 5585	091794081	<a href="#">Editar</a>	<a href="#">Eliminar</a>
3785468-2	Facundo	Cardoso	Aigua 3588	091463558	<a href="#">Editar</a>	<a href="#">Eliminar</a>

© 2023 - My ASP.NET Application

### 3.5 Cliente

Cuentas Vehículos Ventas Alquileres Usuarios

### Venta

Cuentas:

Documento: 4586658-0 _No
--------------------------

Vehículos:

Tipo: Auto _Matrícula: M445
-----------------------------

Fecha:

dd/mm/aaaa
------------

### Lista de vehículos vendidos

© 2023 - My ASP.NET Application

### 3.5 Venta

Ciudades Vehículos Ventas Alquileres Usuarios

### Registro de Alquiler

Clientes:  
Documento: 4506658-0 \_Nombre: 1

Vehículos:  
Tipo: Auto \_Matrícula: MA458855\_1

Fecha:  
dd/mm/aaaa

Días de Alquiler:  
Ingrese los días del alquiler

**Guardar**

### Listado de Alquileres

© 2023 - My ASP.NET Application

### 3.5 Alquiler

Ciudades Vehículos Ventas Alquileres Usuarios

### Agregar un nuevo Usuario

Nombre del Usuario  
Apellido del Usuario  
.....  
Confirmar Contraseña

Permisos:

- Ver Clientes
- Ver Usuario
- Ver Ventas
- Ver Vehículos
- Ver Alquileres

**Guardar**

Número ID Usuario	Username	Nombre	Apellido	Editar	Eliminar
0	Admin			<a href="#">Editar</a>	<a href="#">Eliminar</a>

© 2023 - My ASP.NET Application

### 3.5 Usuario

# Resultados

## Gestión de Usuarios

- Establecimiento eficiente de permisos de usuario, permitiendo asignar funciones específicas a cada usuario.
- Validación de cédula para clientes uruguayos, con funciones de login, registro y administración de usuarios.

## Administración de Vehículos

- Tipos de vehículos diferenciados con atributos específicos para motos, autos y camiones.
- Incorporación obligatoria de imágenes para cada vehículo

## Operaciones de Venta y Alquiler

- Implementación de pantallas para la venta y alquiler de vehículos, con búsqueda, selección de cliente y registro de información relevante.
- Listados de vehículos vendidos y en alquiler, con funcionalidades adicionales como marcado de devolución y verificación de atrasos.

## Interfaz de Usuario

- Desarrollo de una interfaz amigable con menús de consola que facilitan la interacción del usuario.
- Funcionalidades que permiten agregar legisladores, listar cámaras de trabajo, contar legisladores por tipo y realizar acciones específicas.

## Documentación y Seguridad

- Código fuente organizado y documentado.
- Implementación de medidas de seguridad, como validación de datos de entrada.
- Manual de usuario detallado para guiar a los usuarios en el uso efectivo del sistema.

# Conclusión

En el transcurso de este proyecto, hemos trazado un camino sólido hacia el logro de metas significativas que fortalecen la eficacia y solidez de nuestro sistema. La exitosa implementación de funciones clave en áreas cruciales, como la gestión de usuarios, administración de vehículos, operaciones de venta y alquiler, así como el diseño de la interfaz de usuario, refleja un enfoque consciente y centrado en la experiencia del usuario.

Destacamos la gestión de usuarios, donde la asignación eficiente de permisos y la meticulosa validación de cédulas han establecido un control preciso sobre las funciones del sistema, garantizando una atención exclusiva a nuestros clientes uruguayos. En la administración de vehículos, la construcción de una estructura de datos sólida y la incorporación obligatoria de imágenes han enriquecido de manera significativa la representación y manipulación de información vehicular.

Las operaciones de venta y alquiler han sido diseñadas cuidadosamente con una interfaz intuitiva que simplifica la interacción del usuario, facilitando búsquedas eficientes, la selección de clientes y un registro detallado de cada transacción. Los listados de vehículos vendidos y en alquiler, junto con funcionalidades adicionales como el marcado de devolución y la verificación de atrasos, agregan una capa de control y seguimiento invaluable.

La interfaz de usuario, con sus menús de consola amigables y funciones prácticas, ha contribuido de manera significativa a una experiencia de usuario positiva.

## Referencias

1. <https://www.hostingplus.pe/blog/conceptodeideycualessonsuscaracteristicas/>
2. <https://desarrolloweb.com/articulos/clases-objetos-c-sharp>
3. <https://learn.microsoft.com/es-es/dotnet/csharp/programming-guide/classes-and-structures/methods>
4. <https://rockcontent.com/es/blog/bootstrap/>
5. <https://blog.hubspot.es/website/que-es-css#que-es>
6. [https://developer.mozilla.org/es/docs/Learn/Getting\\_started\\_with\\_the\\_web/HTML\\_basics](https://developer.mozilla.org/es/docs/Learn/Getting_started_with_the_web/HTML_basics)
7. [https://estradawebgroup.com/Post/HerenciadeclasesenCunaguiapractica/20665?expand\\_article=1](https://estradawebgroup.com/Post/HerenciadeclasesenCunaguiapractica/20665?expand_article=1)
8. <https://learn.microsoft.com/es-es/dotnet/csharp/fundamentals/types/classes>

