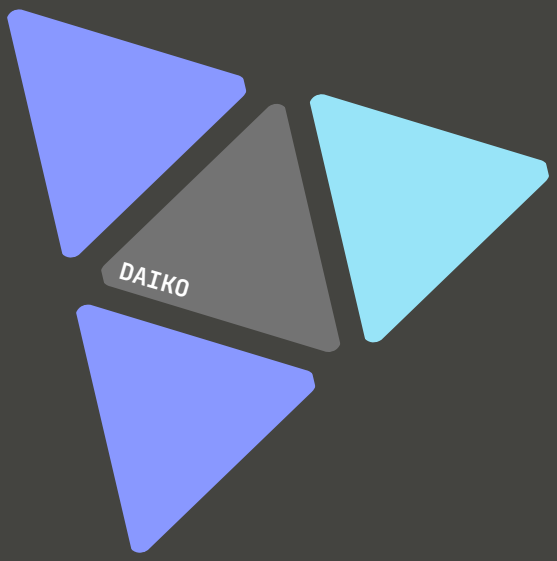
DAIKO SYSTEM

SOFTWARE DE GESTION



Kevin Caballero

Desarrollo de Aplicaciones Multiplataforma
S.E.I.M. Centro de estudios de Informática

ÍNDICE

FASES

IDENTIFICACIÓN DEL PROYECTO

OBJETIVO GENERAL

Se plantea el desarrollo de un software de gestión multiplataforma que sea intuitivo, agradable a la vista y eficiente.

ANTECEDENTES

Con la **primera versión** del software a desarrollar, se cubren necesidades referentes a PYMES tales como: gestión y control de **inventario**, **recepción y compra** de materias, y gestión de **ventas**.

Aunque el sector de los ERP´s esté **creciendo** significativamente en los últimos años y cuente ya con una amplia oferta de productos, nuestro software se diferencia de otros por el **apartado gráfico** y su sencillez de cara al usuario.

REQUISITOS MÍNIMOS

Procesador: Intel Pentium IV
Memoria: 1 GB
Almacenamiento: 10 GB
S.O: Windows 7

*RECOMENDACION
Un equipo únicamente dedicado a gestión del lado servidor con windows server.

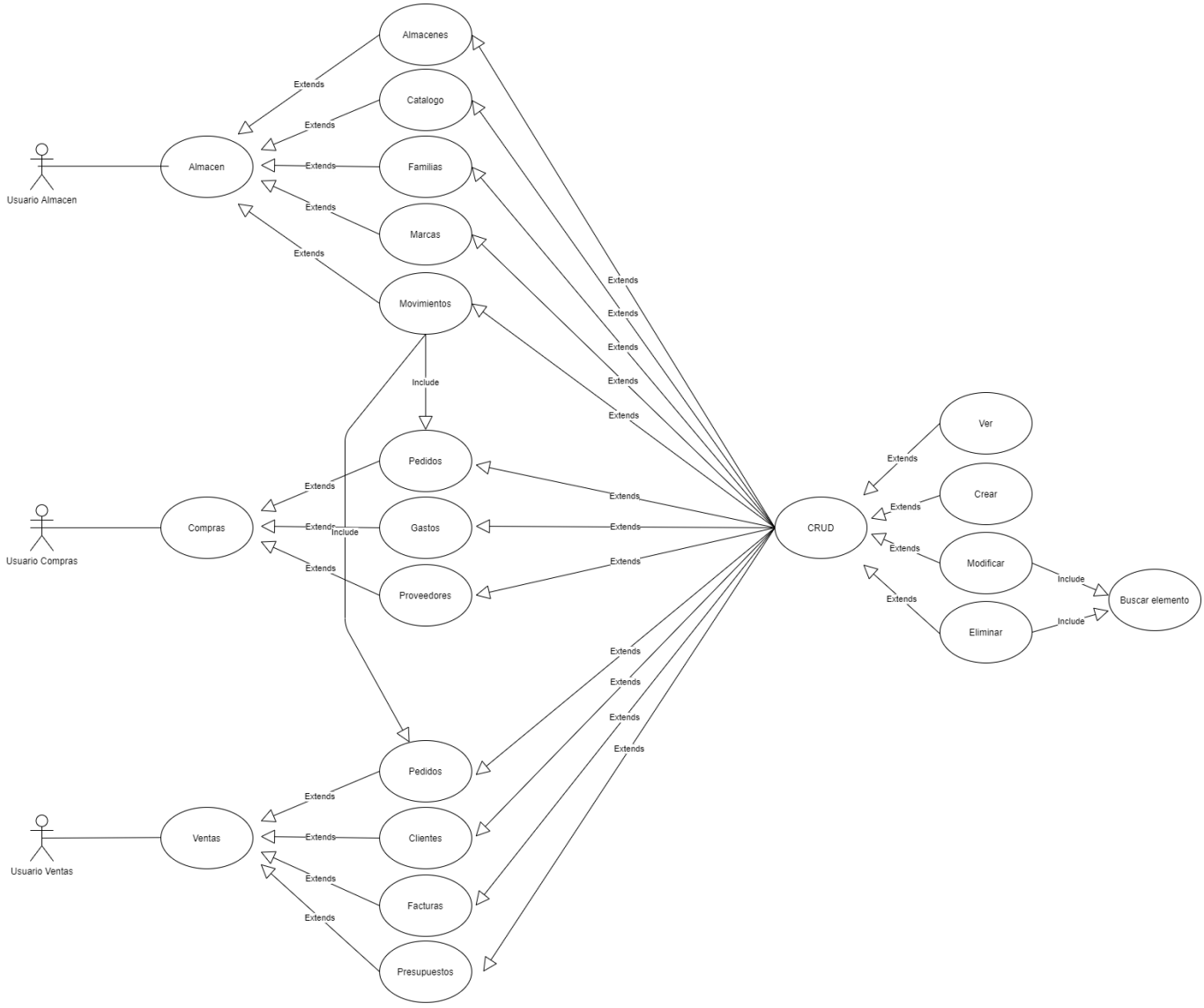
TECNOLOGÍAS

Para la primera versión se ha pensado en implementar tecnologías tales como "NET FRAMEWORK", tanto para la capa de cliente como para la capa servidor y, "SQL SERVER", como motor de base de datos.

Se ha optado por no usar ORM para buscar un plus de velocidad en el rendimiento.

**Para futuras versiones se plantea un stack de tecnologías más actual., como "NODE + EXPRESS" para la capa servidor, "MONGODB" como motor de base de datos y Angular o REACT como capa cliente.*

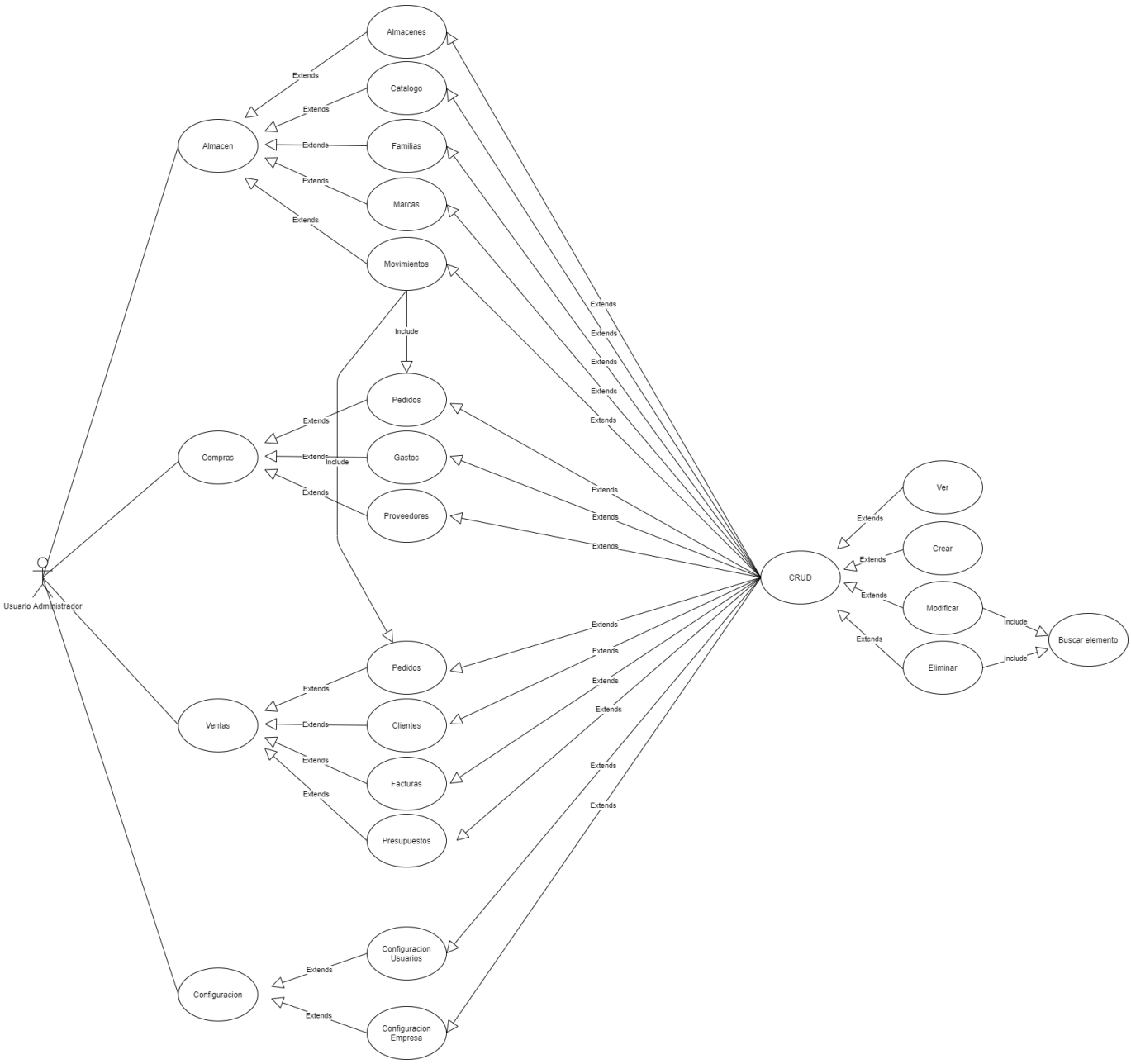
ANÁLISIS



CASOS DE USO USUARIOS NO ADMIN.

Los usuarios **no administradores** tendrán acceso a la parte del software con el mismo nombre de su **sección**, en la que podrán acceder a todas las áreas de dicha parte y en cada una de estas realizar "**CRUD**".

ANÁLISIS



CASOS DE USO USUARIOS ADMIN.

Los usuarios **administradores** tendrán acceso completo a todas las características del sistema, pudiendo modificar datos de **usuarios y datos de la empresa** de ser necesario.

DISEÑO

ARQUITECTURA

La arquitectura de la primera versión consistirá en cuatro capas diferenciadas.

- Capa de interfaz de usuario.
- Capa de lógica de negocio.
- Capa de acceso a datos.
- Capa común.

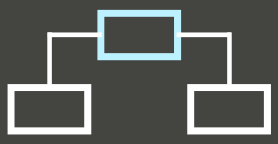


DIAGRAMA DE CLASES

Las cuatro clases principales del proyecto son:

- Daiko_UI: encargada de pintar la interfaz de usuario y gestionar pequeñas lógicas no referentes a la capa de negocio.
- Daiko_UC: encargada de contener los modelos comunes a todo el sistema.
- Daiko_DLL: encargada de gestionar todo lo referente a la lógica de negocio.
- Daiko_DAL: encargada únicamente del acceso a datos, separándose así de la logica de la capa de negocio.

INTERFAZ GRÁFICA

La interfaz gráfica dispondrá de colores asociativos para cada una de las secciones de la empresa.

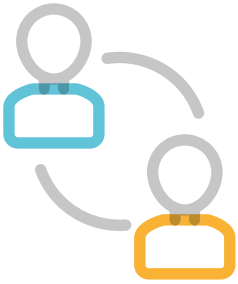
Se han realizado diferentes "mockups" para las principales pantallas de la aplicación.

Se ha procurado que las interfaces sean sencillas y claras a la vez que bonitas y agradables.

ENTIDAD/RELACIÓN

El diagrama de entidad/relación es el proporcionado por "sql server management studio".

De dicho diagrama, en esta primera versión, solo estará funcionando el grupo de tablas de almacén y parte del grupo de general.



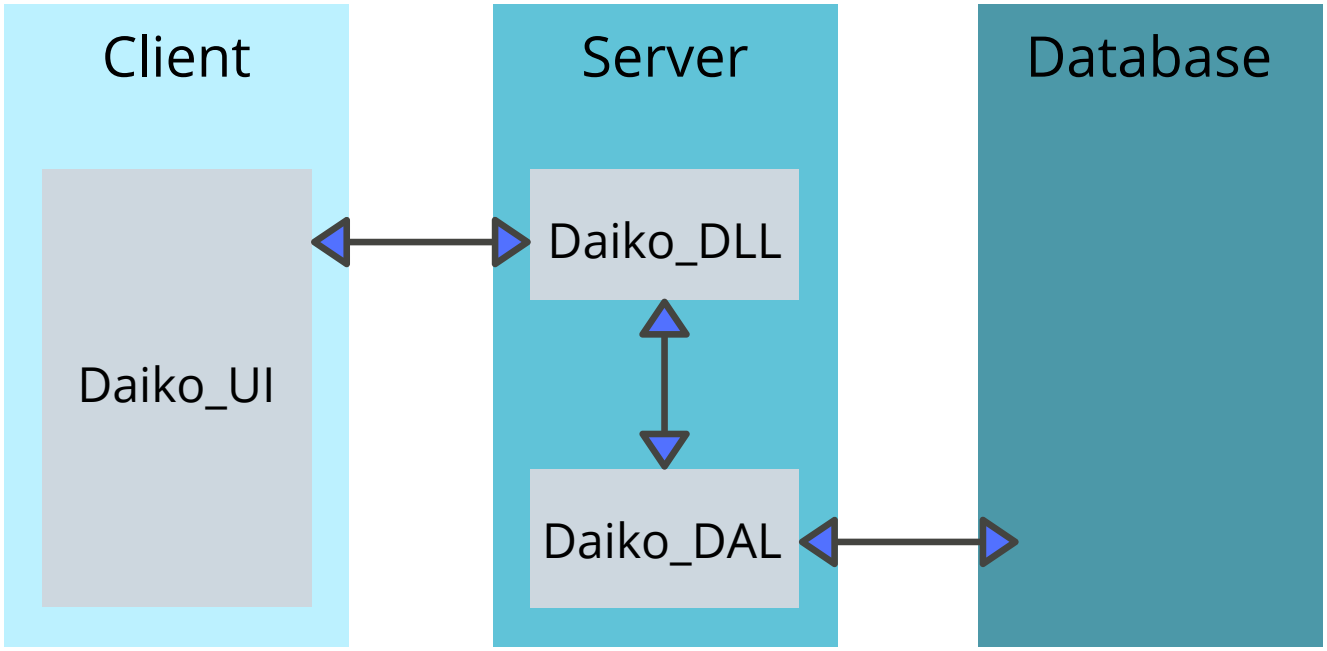
DISEÑO

ARQUITECTURA

Para la primera versión se ha optado por una arquitectura en tres capas interrelacionadas. Separando así la capa de interfaz de la capa de lógica de negocio, y esta a su vez de la capa de acceso a datos.

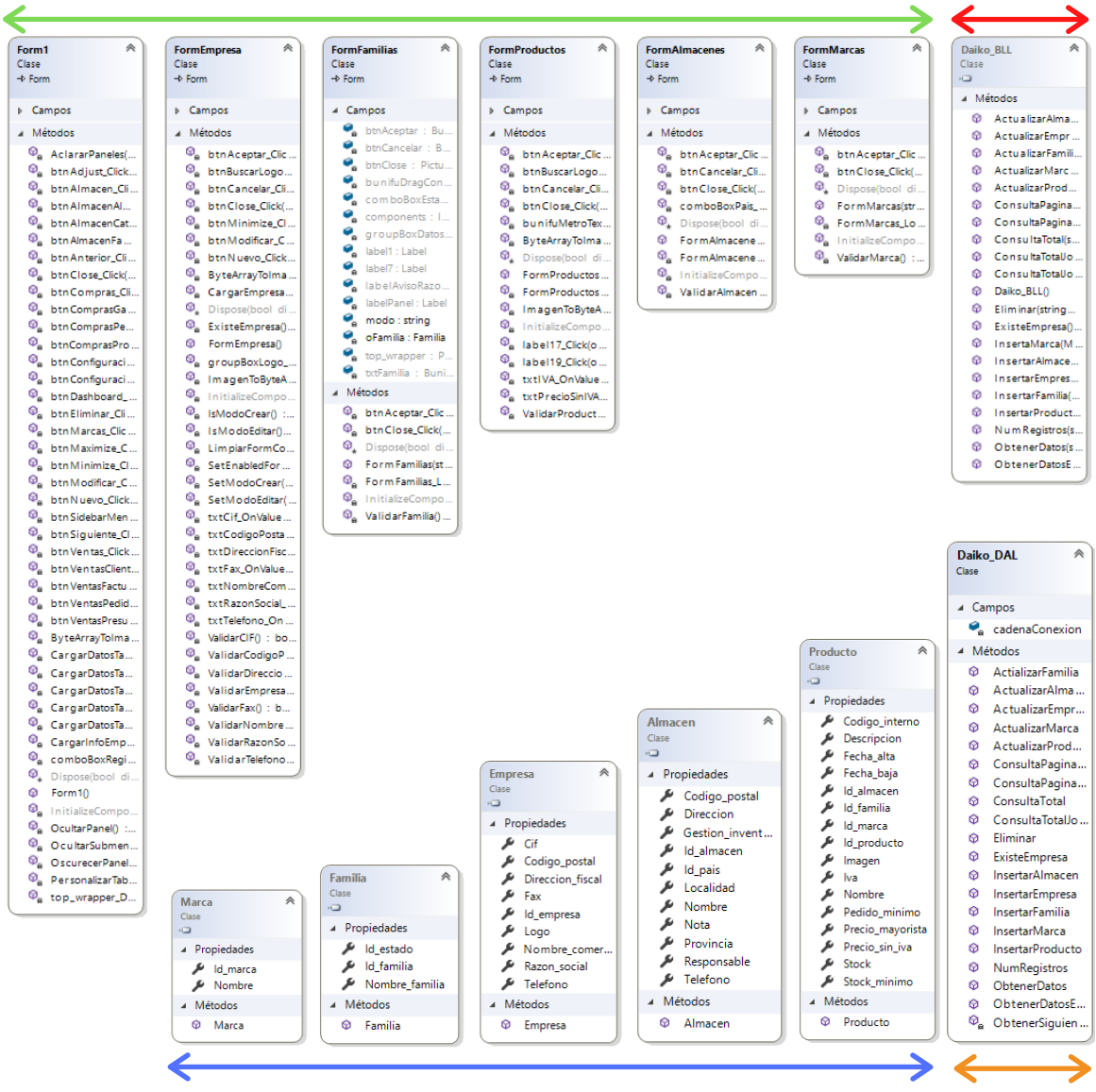
En esta primera versión no tendrá demasiada carga la parte de lógica de negocio pero es la capa que se encargaría de realizar los cálculos para mostrar en los gráficos del "dashboard" entre otras cosas.

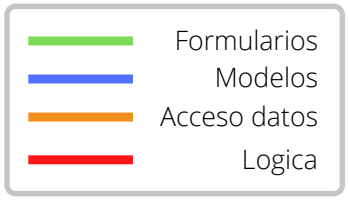
Para futuras versiones se plantea implementar una arquitectura limpia como por ejemplo la arquitectura hexagonal, aplicando los principios SOLID.



DISEÑO

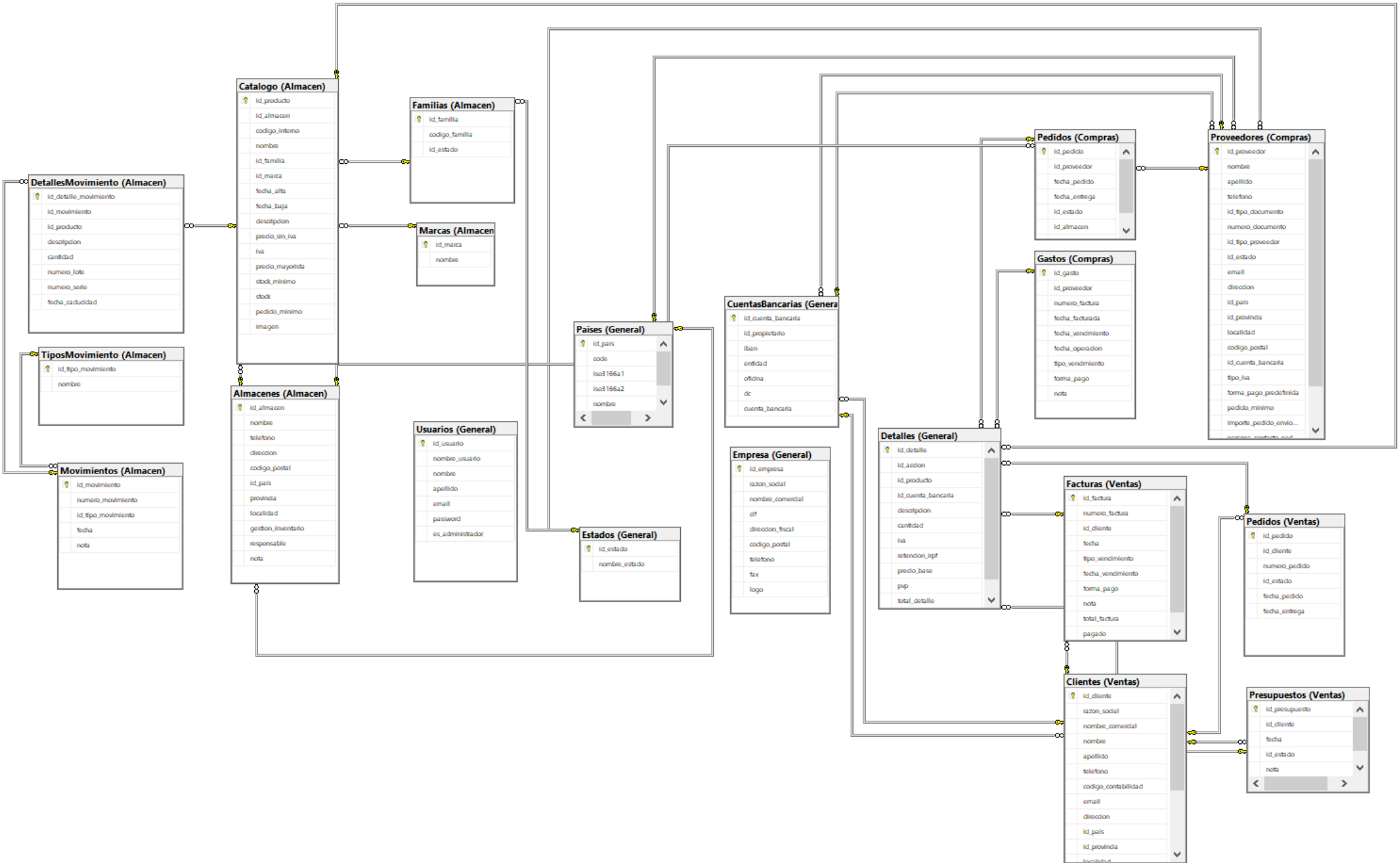
CLASES





En la imagen se pueden diferencias claramente los 4 proyectos que conforman el sistema. La parte de formularios pertenece a Daiko_UI y en ella esta lo relativo a la parte grafica del sistema. La parte de modelos pertenece a Daiko_UC, en este proyecto se encuentra lo común a todos los demás proyectos, como por ejemplo, los modelos. Por ultimo tenemos las clases de Daiko_BLL y Daiko_DAL que hacen referencia a la logica y al acceso a datos.

ENTIDAD/RELACIÓN



MEMORIA TECNICA

PROYECTOS

A continuación se realizara una breve descripción de la utilidad de la proyecto así como de su funcionamiento interno.

DAIKO_DAL

En este proyecto se implementa todo lo relativo al acceso a datos, es decir, tanto la conexión como las consultas que se lanzaran contra la base de datos se almacenan aquí.

Este proyecto se encargaría de devolver únicamente los datos resultantes de la ejecución de la consulta, para que posteriormente sean procesados en la capa de lógica de negocio.

El proyecto Daiko_DAL consta únicamente de una clase poblada de métodos para realizar "CRUD" y otras operaciones necesarias para el sistema.

DAIKO_BLL

En este proyecto se alojaría todo lo relativo a la lógica de negocio y cálculos que no tengan relación ni con el acceso a datos ni con la interfaz de usuario.

Este proyecto consta de una única clase poblada en su mayoría de métodos, ya que el input de datos vendría o bien de la capa de usuario o bien de la capa de datos.

Actualmente esta clase únicamente redirecciona hacia una capa superior, pero seria utilizada para lo anteriormente comentado, por ejemplo, realizar los cálculos referentes a la información que se mostrara en el dashboard.

Esta capa, aunque actualmente se encuentre en local esta pensada para albergarse en un servidor externo y ser accedida desde un cliente, lo único que necesitaría ser modificado es el método de conexión.

**Ante la imposibilidad de depurar en estas capas he implementado un sistema de trazas y excepciones para agilizar así el desarrollo y conocer en cada momento lo que esta pasando con el código.*

MEMORIA TECNICA

DAIKO_UI

En este proyecto se implementa todo lo referente a la capa de presentación, es decir aquella capa mediante la cual el usuario interactua con el sistema.

Consta de un formulario principal en el cual se van desplazando los paneles de su interior para hacer visibles unos u otros, y los formularios de inserción de datos de cada uno de los apartados del menú lateral.

En esta capa se encuentra únicamente la lógica necesaria para el manejo de elementos gráficos, no se realizan cálculos de ningún tipo ni se aplica lógica de negocio en esta capa.

DAIKO_UC

En este proyecto se implementan todos los modelos de datos para que se pueda acceder a ellos desde cualquier parte del sistema. Todos estos modelos se encuentran en este proyecto dentro de una carpeta llamada "Models".

El resto de proyectos del sistema, cuentan con una referencia Daiko_UC por lo anteriormente comentado.

Esta capa carece de lógica, únicamente es un acceso a los modelos que sustituirían a los archivos generados por un ORM como podría ser "Entity framework"

MEMORIA TECNICA

CLASES DAIKO_DAL

A continuación se realizara una breve descripción de la utilidad de las clases de acceso a datos así como de su funcionamiento interno.

DAIKO_DAL.CS

Como hemos comentado anteriormente, esta clase esta poblada únicamente por métodos (los cuales se explican mas adelante) para acceder a datos. Su función principal es obtener los datos en bruto de la base de datos.

La única variable que posee esta clase es el string de la cadena de conexión:

```
private static string cadenaConexion = "Data Source=DESKTOP-G9E2PVR;Initial Catalog=daiko_system;
                                        Integrated Security = True";
```

CLASES DAIKO_BLL

A continuación se realizara una breve descripción de la utilidad de las clases de lógica de negocio así como de su funcionamiento interno.

DAIKO_BLL.CS

Como hemos comentado anteriormente, esta clase esta poblada únicamente por métodos ya que el input de datos procede de las otras capas. Actualmente lo que hace es redireccionar lo que recibirnos en las capas de los extremos del sistema ("Daiko_UI" y "Daiko_DAL").

En ella se llevarían a cabo cálculos relativos a la lógica de negocio gracias a los datos obtenidos tanto en los formularios como en la capa de acceso a datos.

MEMORIA TECNICA

CLASES DAIKO_UC

A continuación se realizara una breve descripción de la utilidad de las clases de modelado de datos así como de su funcionamiento interno.

ALMACEN.CS

Estas son las variables necesarias para la representación de un almacén. Todas ellas son accedidas mediante propiedades publicas.

```
private int _id_almacen;
private string _nombre;
private string _telefono;
private string _direccion;
private string _codigo_postal;
private int _id_pais;
private string _provincia;
private string _localidad;
private string _gestion_inventario;
private string _responsable;
private string _nota;
```

EMPRESA.CS

Estas son las variables necesarias para la representación de una empresa. Todas ellas son accedidas mediante propiedades publicas.

```
private int _id_empresa;
private string _razon_social;
private string _nombre_comercial;
private string _cif;
private string _direccion_fiscal;
private string _codigo_postal;
private string _telefono;
private string _fax;
private byte[] _logo;
```

MEMORIA TECNICA

FAMILIA.CS

Estas son las variables necesarias para la representación de una familia. Todas ellas son accedidas mediante propiedades publicas.

```
private int _id_familia;
private string _nombre_familia;
private string _id_estado;
```

MARCA.CS

Estas son las variables necesarias para la representación de una marca. Todas ellas son accedidas mediante propiedades publicas.

```
private int _id_marca;
private string _nombre;
```

PRODUCTO.CS

Estas son las variables necesarias para la representación de un producto. Todas ellas son accedidas mediante propiedades publicas.

```
private int _id_producto;
private int _id_almacen;
private string _codigo_interno;
private string _nombre;
private int _id_familia;
private int _id_marca;
private DateTime _fecha_alta;
private DateTime _fecha_baja;
private string _descripcion;
private decimal _precio_sin_iva;
private int _iva;
private decimal _precio_mayorista;
private int _stock_minimo;
private int _stock;
private int _pedido_minimo;
private byte[] _imagen;
```

MEMORIA TECNICA

CLASES DAIKO_UI

A continuación se realizara una breve descripción de la utilidad de las clases de interfaz gráfica así como de su funcionamiento interno.

FORM1.CS

Es el formulario principal de la aplicación. En el se realizan diversas funciones para manejar el comportamiento de la tabla y de los distintos componentes gráficos con los que interactuara el usuario.

Con la intención de no mostrar demasiados formularios, para no sobrecargar al usuario, y que sea lo mas parecido posible a una "single page application", se decidió basar el formulario principal en paneles, que se mostraran y ocultaran en base a las acciones del usuario.

Por defecto la tabla de datos en la que se plasman los elementos del submenú seleccionado mostrara 25 registros (si los hubiera).

Estas son las variables necesarias para el funcionamiento de diversas funciones relacionadas con el manejo de datos sobre la tabla. Todas ellas son accedidas mediante propiedades publicas. La explicación y la razón de ser de las variables del formulario se explicara en el apartado de funciones.

```
int maximoPaginas;
int totalRegistros;
int registrosPorPagina = 25;
int paginaActual = 1;
int offset;
string parametrosSelect;

string mod1,mod2,mod3;
string tab1,tab2,tab3;
string
```

MEMORIA TECNICA

FORMALMACENES.CS

Es el formulario en el que el usuario realizara el "CRUD" sobre los almacenes. Este esta compuesto por diferentes controles como "dropdowns", "cajas de texto", etc. para que el usuario introduzca los datos, sea bien, para crear un almacén o para modificar uno existente.

Al clicar sobre una linea de la tabla y pulsar el botón eliminar se nos abrirá este mismo formulario con los datos del elemento seleccionado para que antes de borrar comprobemos todos los datos y posteriormente decidir si queremos eliminarlo o no.

Los formularios hijo del formulario principal, se crean con dos parámetros. Estos parámetros le sirven al sistema para saber en que modo debe abrir el formulario y en caso de que sea editar o eliminar, se le pasara un objeto con el almacén que debe ser mostrado sobre los campos del formulario.

```
string modo;
Daiko_UC.Models.Almacen oAlmacen;
```

FORMEMPRESA.CS

En este formulario, el usuario podrá dar de alta a una única empresa y podrá modificarla pero no podrá eliminarla, esto se debe a que no tendría sentido realizar registros de acciones sin una empresa respaldando dichas acciones.

Este formulario se abre de manera diferente al resto, no tiene un modo de apertura, si no que al iniciarse el formulario se comprueba si existe ya una empresa en la base de datos. Si ya existe una empresa automáticamente se deshabilitara el botón de crear empresa y únicamente quedara activo el botón de modificar.

```
string nombreArchivo;
string urlArchivo;
byte[] logoBytes = Array.Empty<byte>();
```

FORMFAMILIAS.CS

Es el formulario en el que el usuario realizara el "CRUD" sobre las familias. Este esta compuesto por diferentes controles como "dropdowns", "cajas de texto", etc. para que el usuario introduzca los datos, sea bien, para crear una familia o para modificar una existente.

Los formularios hijo del formulario principal, se crean con dos parámetros.

```
string modo;
Daiko_UC.Models.Familia oFamilia;
```

MEMORIA TECNICA

FORMMARCAS.CS

Es el formulario en el que el usuario realizara el "CRUD" sobre las marcas. Este esta compuesto por diferentes controles como "dropdowns", "cajas de texto", etc. para que el usuario introduzca los datos, sea bien, para crear una marca o para modificar una existente.

Los formularios hijo del formulario principal, se crean con dos parámetros.

```
string modo;
Daiko_UC.Models.Familia oFamilia;
```

FORMPRODUCTOS.CS

Es el formulario en el que el usuario realizara el "CRUD" sobre los productos. Este esta compuesto por diferentes controles como "dropdowns", "cajas de texto", etc. para que el usuario introduzca los datos, sea bien, para crear un producto o para modificar uno existente.

Los formularios hijo del formulario principal, se crean con dos parámetros. Este ademas necesita de unas variables para gestionar la imagen del producto. Se deberán almacenar tanto el nombre del archivo como la uri, ademas del array de bytes de la propia imagen.

Para guardar la imagen en la base de datos he optado por guardar el array de bytes en lugar de guardarlo como "BLOB", por lo tanto debe convertirse a ese tipo la imagen introducida por el usuario.

Al contrario cuando lo que se quiere es carbar la imagen en el "picturebox" desde la base de datos, lo que hacemos es convertir dicho array de bytes a "Bitmap".

```
string modo;
Daiko_UC.Models.Familia oFamilia;

string nombreArchivo;
string urlArchivo;
byte[] logoBytes = Array.Empty<byte>();
```

MEMORIA TECNICA

METODOS DAIKO_DAL.CS

A continuación se realizara una breve descripción de la utilidad de los métodos referentes al acceso de datos así como de su funcionamiento interno.

```
public int InsertarX(TipoX.nuevoX)
```

Función encargada de introducir en la base de datos el elemento creado en el respectivo formulario.
Crea un objeto del tipo SqlConnection gracias a la variable string cadenaConexion definida como miembro de la clase, también crea un objeto de tipo SqlCommand.
Se asigna el texto de la consulta al CommandText del objeto SqlCommand con sus respectivos parámetros.
Se le asigna el objeto SqlConnection al Connection del objeto SqlCommand.
Una vez hecho esto podemos abrir la conexión y ejecutar la "INSERT". Si todo ha ido como debería la función devolverá 1, en caso contrario, devolverá -1.

```
public int ActualizarX(TypoX.XActualizado)
```

Función encargada de actualizar en la base de datos el elemento seleccionado en el respectivo formulario.
Crea un objeto del tipo SqlConnection gracias a la variable string cadenaConexion definida como miembro de la clase, también crea un objeto de tipo SqlCommand.
Se asigna el texto de la consulta al CommandText del objeto SqlCommand con sus respectivos parámetros.
Se le asigna el objeto SqlConnection al Connection del objeto SqlCommand.
Una vez hecho esto podemos abrir la conexión y ejecutar la "UPDATE". Si todo ha ido como debería la función devolverá 1, en caso contrario, devolverá -1.

```
public int Eliminar(string modulo, string
tabla,string nombreColumnaClave, int id)
```

Función encargada de eliminar en la base de datos el elemento seleccionado en el respectivo formulario.
Crea un objeto del tipo SqlConnection gracias a la variable string cadenaConexion definida como miembro de la clase, también crea un objeto de tipo SqlCommand.
Se asigna el texto de la consulta al CommandText del objeto SqlCommand con sus respectivos parámetros.
Se le asigna el objeto SqlConnection al Connection del objeto SqlCommand.
Una vez hecho esto podemos abrir la conexión y ejecutar la "DELETE". Si todo ha ido como debería la función devolverá 1, en caso contrario, devolverá -1.

MEMORIA TECNICA

```
public DataTable ConsultaTotal(string modulo,
string tabla, string parametrosSelect)
```

Función encargada de traer de vuelta todo el volumen de datos de una tabla en formato "datatable".
Crea un objeto del tipo SqlConnection gracias a la variable string cadenaConexion definida como miembro de la clase, también crea un objeto de tipo SqlCommand y un objeto DataTable.
Se asigna el texto de la consulta al CommandText del objeto SqlCommand con sus respectivos parámetros.
Se le asigna el objeto SqlConnection al Connection del objeto SqlCommand.
Una vez hecho esto podemos abrir la conexión y haciendo uso de la directiva using de SqlDataAdapter sobre el que se volcaran los datos para posteriormente poblar el datatable.
Si todo ha ido como debería la función devolverá un datatable poblado, en caso contrario, devolverá null.

```
public DataTable ConsultaTotalJoin(string modulo1,
string tabla1,string campoRelacionado1, string
modulo2, string tabla2, string campoRelacionado2,
string parametrosSelect)
```

Función encargada de traer de vuelta todo el volumen de datos de la join entre dos tablas en formato "datatable".
Crea un objeto del tipo SqlConnection gracias a la variable string cadenaConexion definida como miembro de la clase, también crea un objeto de tipo SqlCommand y un objeto DataTable.
Se asigna el texto de la consulta al CommandText del objeto SqlCommand con sus respectivos parámetros.
Se le asigna el objeto SqlConnection al Connection del objeto SqlCommand.
Una vez hecho esto podemos abrir la conexión y haciendo uso de la directiva using de SqlDataAdapter sobre el que se volcaran los datos para posteriormente poblar el datatable.
Si todo ha ido como debería la función devolverá un datatable poblado, en caso contrario, devolverá null.

```
public DataTable ConsultaTotalJoin(string modulo1,
string tabla1, string campoRelacionado1, string
modulo2, string tabla2, string campoRelacionado2,
string modulo3, string tabla3, string
campoRelacionado3, string parametrosSelect)
```

Función encargada de traer de vuelta todo el volumen de datos de la join entre tres tablas en formato "datatable".
Crea un objeto del tipo SqlConnection gracias a la variable string cadenaConexion definida como miembro de la clase, también crea un objeto de tipo SqlCommand y un objeto DataTable.
Se asigna el texto de la consulta al CommandText del objeto SqlCommand con sus respectivos parámetros.
Se le asigna el objeto SqlConnection al Connection del objeto SqlCommand.
Una vez hecho esto podemos abrir la conexión y haciendo uso de la directiva using de SqlDataAdapter sobre el que se volcaran los datos para posteriormente poblar el datatable.
Si todo ha ido como debería la función devolverá un datatable poblado, en caso contrario, devolverá null.

```
public DataTable ConsultaPaginada(string modulo, string tabla, string parametrosSelect, int offset, int registrosPagina, string orden, int numColumnaOrdenacion)
```

Función encargada de traer de vuelta un conjunto limitado de datos de una tabla, en formato "datatable".
La cantidad de registros a traer desde la base de datos, viene definida por el parámetro "registrosPagina", por otra parte , el numero de registro desde el que vamos a avanzar en la búsqueda, viene definido por el parámetro "offset".
De esta manera avanzamos desde el numero de registro, tantos filas como se hayan establecido en el parámetro "registrosPagina", ademas se puede establecer el criterio de ordenación y la columna por la que se desea ordenar.
Se crea un objeto del tipo SqlConnection gracias a la variable string cadenaConexion definida como miembro de la clase, también crea un objeto de tipo SqlCommand y un objeto DataTable.
Se asigna el texto de la consulta al CommandText del objeto SqlCommand con sus respectivos parámetros.
Se le asigna el objeto SqlConnection al Connection del objeto SqlCommand.
Una vez hecho esto podemos abrir la conexión y haciendo uso de la directiva using de SqlDataAdapter sobre el que se volcaran los datos para posteriormente poblar el datatable.
Si todo ha ido como debería la función devolverá un datatable poblado, en caso contrario, devolverá null.

```
public DataTable ConsultaPaginadaJoin(string modulo1, string tabla1, string campoRelacionado1, string modulo2, string tabla2, string campoRelacionado2, string parametrosSelect, int offset, int registrosPagina, string orden, int numColumnaOrdenacion)
```

Función encargada de traer de vuelta un conjunto limitado de datos de la join entre dos tablas, en formato "datatable".
La cantidad de registros a traer desde la base de datos, viene definida por el parámetro "registrosPagina", por otra parte , el numero de registro desde el que vamos a avanzar en la búsqueda, viene definido por el parámetro "offset".
De esta manera avanzamos desde el numero de registro, tantos filas como se hayan establecido en el parámetro "registrosPagina", ademas se puede establecer el criterio de ordenación y la columna por la que se desea ordenar.
Se crea un objeto del tipo SqlConnection gracias a la variable string cadenaConexion definida como miembro de la clase, también crea un objeto de tipo SqlCommand y un objeto DataTable.
Se asigna el texto de la consulta al CommandText del objeto SqlCommand con sus respectivos parámetros.
Se le asigna el objeto SqlConnection al Connection del objeto SqlCommand.
Una vez hecho esto podemos abrir la conexión y haciendo uso de la directiva using de SqlDataAdapter sobre el que se volcaran los datos para posteriormente poblar el datatable.
Si todo ha ido como debería la función devolverá un datatable poblado, en caso contrario, devolverá null.

```
public DataTable ConsultaPaginadaJoin(string modulo1, string tabla1, string campoRelacionado1, string modulo2, string tabla2, string campoRelacionado2, string modulo3, string tabla3, string campoRelacionado3, string parametrosSelect, int offset, int registrosPagina, string orden, int numColumnaOrdenacion)
```

Función encargada de traer de vuelta un conjunto limitado de datos de la join entre tres tabla, en formato "datatable".
La cantidad de registros a traer desde la base de datos, viene definida por el parámetro "registrosPagina", por otra parte , el numero de registro desde el que vamos a avanzar en la búsqueda, viene definido por el parámetro "offset".
De esta manera avanzamos desde el numero de registro, tantos filas como se hayan establecido en el parámetro "registrosPagina", ademas se puede establecer el criterio de ordenación y la columna por la que se desea ordenar.
Se crea un objeto del tipo SqlConnection gracias a la variable string cadenaConexion definida como miembro de la clase, también crea un objeto de tipo SqlCommand y un objeto DataTable.
Se asigna el texto de la consulta al CommandText del objeto SqlCommand con sus respectivos parámetros.
Se le asigna el objeto SqlConnection al Connection del objeto SqlCommand.
Una vez hecho esto podemos abrir la conexión y haciendo uso de la directiva using de SqlDataAdapter sobre el que se volcaran los datos para posteriormente poblar el datatable.
Si todo ha ido como debería la función devolverá un datatable poblado, en caso contrario, devolverá null.

```
public int NumRegistros(string modulo, string tabla)
```

Función encargada de traer de vuelta el numero de registros existentes en una tabla.
Crea un objeto del tipo SqlConnection gracias a la variable string cadenaConexion definida como miembro de la clase, también crea un objeto de tipo SqlCommand.
Se asigna el texto de la consulta al CommandText del objeto SqlCommand con sus respectivos parámetros.
Se le asigna el objeto SqlConnection al Connection del objeto SqlCommand.
Una vez hecho esto podemos abrir la conexión y ejecutar la consulta escalar. Si todo ha ido como debería, la función devolverá el numero de registros, en caso contrario, devolverá 0.

```
private int ObtenerSiguienteID(string modulo, string tabla, string nombreColumnaClave)
```

Función encargada de traer de vuelta el numero del ultimo registro de una tabla añadiéndole uno.
Crea un objeto del tipo SqlConnection gracias a la variable string cadenaConexion definida como miembro de la clase, también crea un objeto de tipo SqlCommand.
Se asigna el texto de la consulta al CommandText del objeto SqlCommand con sus respectivos parámetros.
Se le asigna el objeto SqlConnection al Connection del objeto SqlCommand.
Una vez hecho esto podemos abrir la conexión y ejecutar la consulta escalar. Si la ejecución de la escalar es null, significa que la tabla esta vacía y por lo tanto el siguiente ID seria el 1, por el contrario si no esta vacía devolvería lo anteriormente mencionado.

MEMORIA TECNICA

```
public object ObtenerDatos(string modulo, string
tabla, string parametrosSelect, string
nombreColumnaClave, int id)
```

Función encargada de traer de vuelta los datos de un único elemento de la tabla, en formato de objeto genérico para su posterior conversión al tipo necesario.
Se crea un objeto del tipo SqlConnection gracias a la variable string cadenaConexion definida como miembro de la clase, también crea un objeto de tipo SqlCommand y un objeto DataReader.
Se asigna el texto de la consulta al CommandText del objeto SqlCommand con sus respectivos parámetros.
Se le asigna el objeto SqlConnection al Connection del objeto SqlCommand.
Una vez hecho esto podemos abrir la conexión y volcar los datos de la consulta sobre el datareader. Si el datareader tiene filas, dependiendo del nombre de la tabla que nos hayan pasado como parámetro, ejecutara la función Read() del datareader de una manera u otra, teniendo en cuenta las variables del tipo de dato al que posteriormente se vaya a convertir.

```
public Daiko_UC.Models.Empresa
ObtenerDatosEmpresa()
```

Función encargada de traer de vuelta los datos de la empresa almacenada en la base de datos, en un objeto de tipo Empresa.
Se crea un objeto del tipo SqlConnection gracias a la variable string cadenaConexion definida como miembro de la clase, también crea un objeto de tipo SqlCommand y un objeto DataReader.
Se asigna el texto de la consulta al CommandText del objeto SqlCommand con sus respectivos parámetros.
Se le asigna el objeto SqlConnection al Connection del objeto SqlCommand.
Una vez hecho esto podemos abrir la conexión y volcar los datos de la consulta sobre el datareader. Si el datareader tiene filas, ejecutamos la función Read() del datareader para crear un objeto de tipo Empresa con todos los datos recibidos.

```
public bool ExisteEmpresa()
```

Función encargada de comprobar que en la base de datos, en la tabla empresa, haya un elemento.
Crea un objeto del tipo SqlConnection gracias a la variable string cadenaConexion definida como miembro de la clase, también crea un objeto de tipo SqlCommand.
Se asigna el texto de la consulta al CommandText del objeto SqlCommand con sus respectivos parámetros.
Se le asigna el objeto SqlConnection al Connection del objeto SqlCommand.
Una vez hecho esto podemos abrir la conexión y ejecutar la consulta escalar". Si el resultado de la consulta es menor que uno, retornara false, de lo contrario retornara true.

MEMORIA TECNICA

METODOS FORM1.CS

A continuación se realizara una breve descripción de la utilidad de los métodos referentes al formulario principal, así como de su funcionamiento interno.

```
private void btnMinimize_Click(object sender,
EventArgs e)
```

Función encargada de minimar el formulario, cambiando el FormWindowState a Minimized.

```
private void btnAdjust_Click(object sender,
EventArgs e)
```

Función encargada de ajustar el tamaño de formulario, cambiando el FormWindowState a normal. También hace visible el botón de maximizar y se hace invisible a si mismo.

```
private void btnMaximize_Click(object sender,
EventArgs e)
```

Función encargada de maximizar el formulario, cambiando el FormWindowState a Maximized. También hace visible el botón de ajustar y se hace invisible a si mismo.

```
private void btnClose_Click(object sender,
EventArgs e)
```

Función encargada de cerrar el formulario.

MEMORIA TECNICA

```
private void btnSidebarMenu_Click(object sender,
EventArgs e)
```

Función encargada de encojer y agrandar el menú lateral al hacer clic.

```
private void btnDashboard_Click(object sender,
EventArgs e)
```

Función encargada de traer al frente y hacer visible el panel del Dashboard.

```
private void btnAlmacen_Click(object sender,
EventArgs e)
```

Función encargada de desplegar y replegar el submenu de elementos de Almacén en el sidebar.

```
private void btnAlmacenAlmacenes_Click(object
sender, EventArgs e)
```

Función encargada de modificar los labels de para que se muestre en ellos tanto el menu, como el submenu seleccionados. Hace visible el panel de la tabla y lo trae al frente. Modifica los colores y carga los datos haciendo una consulta paginada.

```
private void btnAlmacenCatalogo_Click(object
sender, EventArgs e)
```

Función encargada de modificar los labels de para que se muestre en ellos tanto el menu, como el submenu seleccionados. Hace visible el panel de la tabla y lo trae al frente. Modifica los colores y carga los datos haciendo una consulta paginada.

```
private void btnAlmacenFamilias_Click(object sender, EventArgs e)
```

Función encargada de modificar los labels de para que se muestre en ellos tanto el menu, como el submenu seleccionados. Hace visible el panel de la tabla y lo trae al frente. Modifica los colores y carga los datos haciendo una consulta paginada.

```
private void btnAlmacenMarcas_Click(object sender, EventArgs e)
```

Función encargada de modificar los labels de para que se muestre en ellos tanto el menu, como el submenu seleccionados. Hace visible el panel de la tabla y lo trae al frente. Modifica los colores y carga los datos haciendo una consulta paginada.

```
private void btnCompras_Click(object sender, EventArgs e)
```

Función encargada de desplegar y replegar el submenu de elementos de Compras en el sidebar.

```
private void btnComprasGastos_Click(object sender, EventArgs e)
```

Función encargada de modificar los labels de para que se muestre en ellos tanto el menú, como el submenu seleccionados. Hace visible el panel de la tabla y lo trae al frente. Modifica los colores y cargaria los datos haciendo una consulta paginada.

```
private void btnComprasPedidos_Click(object sender, EventArgs e)
```

Función encargada de modificar los labels de para que se muestre en ellos tanto el menú, como el submenu seleccionados. Hace visible el panel de la tabla y lo trae al frente. Modifica los colores y cargaria los datos haciendo una consulta paginada.

```
private void btnComprasProveedores_Click(object sender, EventArgs e)
```

Función encargada de modificar los labels de para que se muestre en ellos tanto el menú, como el submenu seleccionados. Hace visible el panel de la tabla y lo trae al frente. Modifica los colores y cargaría los datos haciendo una consulta paginada.

```
private void btnVentas_Click(object sender, EventArgs e)
```

Función encargada de desplegar y replegar el submenu de elementos de Ventas en el sidebar.

```
private void btnVentasClientes_Click(object sender, EventArgs e)
```

Función encargada de modificar los labels de para que se muestre en ellos tanto el menú, como el submenu seleccionados. Hace visible el panel de la tabla y lo trae al frente. Modifica los colores y cargaría los datos haciendo una consulta paginada.

```
private void btnVentasFacturas_Click(object sender, EventArgs e)
```

Función encargada de modificar los labels de para que se muestre en ellos tanto el menú, como el submenu seleccionados. Hace visible el panel de la tabla y lo trae al frente. Modifica los colores y cargaría los datos haciendo una consulta paginada.

```
private void btnVentasPedidos_Click(object sender, EventArgs e)
```

Función encargada de modificar los labels de para que se muestre en ellos tanto el menú, como el submenu seleccionados. Hace visible el panel de la tabla y lo trae al frente. Modifica los colores y cargaría los datos haciendo una consulta paginada.

```
private void btnVentasPresupuestos_Click(object sender, EventArgs e)
```

Función encargada de modificar los labels de para que se muestre en ellos tanto el menú, como el submenu seleccionados. Hace visible el panel de la tabla y lo trae al frente. Modifica los colores y cargaría los datos haciendo una consulta paginada.

```
private void btnConfiguracion_Click(object sender, EventArgs e)
```

Función encargada de hacer visible y traer al frente el panel de configuración.

```
private void btnConfiguracionEmpresa_Click(object sender, EventArgs e)
```

Función encargada de mostrar de manera modal FormEmpresa.cs

```
private void btnSiguiente_Click(object sender, EventArgs e)
```

Función encargada de mostrar los siguientes registros de la tabla. Se modifica el valor de la variable paginaActual y se incrementa en uno. También suma a la variable offset el valor de registrosPagina y ejecuta la función CargarDatosTabla(), realizando así una consulta paginada con los datos establecidos en las variables de clase.

```
private void btnSiguiente_Click(object sender, EventArgs e)
```

Función encargada de mostrar los anteriores registros de la tabla. Se modifica el valor de la variable paginaActual y se reduce en uno. También resta a la variable offset el valor de registrosPagina y ejecuta la función CargarDatosTabla(), realizando así una consulta paginada con los datos establecidos en las variables de clase.

```
private void btnNuevo_Click(object sender,
EventArgs e)
```

Función encargada de abrir el formulario pertinente en modo crear, para que a través de el podamos añadir nuevos elementos a la base de datos.
Se oscurece el back y se abre el formulario pertinente pasandole "crear" como string en el parámetro modo y null en el objeto.
Una vez se cierre el formulario, se volveran a cargar los datos actualizados en la tabla.

```
private void btnModificar_Click(object sender,
EventArgs e)
```

Función encargada de abrir el formulario pertinente en modo editar, para que a través de el podamos modificar elementos existentes en la base de datos.
Se comprueba que hay almenos una fila de la tabla seleccionada y se obtiene el id de dicha fila, para con el generar un objeto.
Se oscurece el back y se abre el formulario pertinente pasandole "modificar" como string en el parámetro modo y el objeto que hemos creado gracias al id de la fila.
Una vez se cierre el formulario, se volverán a cargar los datos actualizados en la tabla.

```
private void btnEliminar_Click(object sender,
EventArgs e)
```

Función encargada de abrir el formulario pertinente en modo eliminar, para que a través de el podamos eliminar elementos existentes en la base de datos.
Se comprueba que hay almenos una fila de la tabla seleccionada y se obtiene el id de dicha fila, para con el generar un objeto.
Se oscurece el back y se abre el formulario pertinente pasandole "modificar" como string en el parámetro modo y el objeto que hemos creado gracias al id de la fila. El formulario mostrara los datos de dicho objeto para que el usuario pueda comprobar si realmente es el elemento que desea eliminar.
Una vez se cierre el formulario, se volverán a cargar los datos actualizados en la tabla.

```
private void top_wrapper_DoubleClick(object
sender, EventArgs e)
```

Función encargada de maximizar o ajustar el tamaño del formulario al hacer doble click sobre el marco superior de la ventana.

```
private void OcultarSubmenus(object sender,
EventArgs e)
```

Función encargada de ocultar los elementos referentes a los diferentes submenus del sidebar. Esto se logra modificando el tamaño de los elementos padre.

```
private void OcultarPanel(object sender, EventArgs
e)
```

Función encargada de ocultar todos los paneles cambiando su visibilidad a false.

```
private void OscurecerPaneles(object sender,
EventArgs e)
```

Función encargada de oscurecer los paneles cuando un elemento se muestra sobre ellos. Lo que hace es cambiar el backcolor de los paneles a un tono mas oscuro.

```
private void AclararPaneles(object sender,
EventArgs e)
```

Función encargada de aclararlos paneles cuando un elemento que se mostraba sobre ellos ha dejado de hacerlo. Lo que hace es cambiar el backcolor de los paneles a un tono mas claro.

```
private void CargarInfoEmpresa(object sender,
EventArgs e)
```

Función encargada de aplicar personalización al elemento situado sobre el sidebar. En el caso de que exista empresa, se obtiene los datos de la misma y se pone su nombre en el label de la parte superior del sidebar. Si la empresa tiene logo, convertirá el byteArray recibido desde la base de datos a una imagen y la cargara a su vez en el picturebox de la parte superior del sidebar.

```
private Image ByteArrayToImagen(object sender,
EventArgs e)
```

Función encargada de convertir el byteArray que recibimos desde la base de datos en un bitmap.

```
private void CargarDatosTabla(string modulo,
string tabla, string parametrosSelect)
```

Función encargada de cargar en la tabla el resultado de una consulta no paginada.
Se establece como fuente de datos el resultado de la funcion ConsultaTotal() de la capa de lógica de negocio pasandole el modulo o seccion, la tabla y los parametros de la select..
Se modifica el ancho de la primera columna de la tabla y se obtiene el numero de registros de la tabla de la base de datos. Se activan y desactivan los botones de navegación de la tabla en base a la cantidad de registros por mostrar y se modifica el mensaje informativo situado en la parte inferior de la tabla.

```
private void CargarDatosTabla(string modulo1,
string tabla1, string campoRelacionado1, string
modulo2, string tabla2, string campoRelacionado2,
string parametrosSelect)
```

Función encargada de cargar en la tabla el resultado de una consulta no paginada con join entre dos tablas .
Se establece como fuente de datos el resultado de la función ConsultaTotal() de la capa de lógica de negocio pasandole los módulos o secciones, las tablas, los campos relacionados y los parámetros de la select..
Se modifica el ancho de la primera columna de la tabla y se obtiene el numero de registros de la tabla de la base de datos. Se activan y desactivan los botones de navegación de la tabla en base a la cantidad de registros por mostrar y se modifica el mensaje informativo situado en la parte inferior de la tabla.

```
private void CargarDatosTabla(string modulo1, string tabla1,
string campoRelacionado1, string modulo2, string tabla2, string
campoRelacionado2,string modulo3, string tabla3, string
campoRelacionado3, string parametrosSelect)
```

Función encargada de cargar en la tabla el resultado de una consulta no paginada con join entre tres tablas .
Se establece como fuente de datos el resultado de la función ConsultaTotal() de la capa de lógica de negocio pasandole los módulos o secciones, las tablas, los campos relacionados y los parámetros de la select..
Se modifica el ancho de la primera columna de la tabla y se obtiene el numero de registros de la tabla de la base de datos. Se activan y desactivan los botones de navegación de la tabla en base a la cantidad de registros por mostrar y se modifica el mensaje informativo situado en la parte inferior de la tabla.

```
private void CargarDatosTabla(string modulo,
string tabla, string parametrosSelect, int offset,
int registrosPorPagina, string criterioOrdenacion,
int posicionColumnaPK)
```

Función encargada de cargar en la tabla el resultado de una consulta paginada .
Se establece como fuente de datos el resultado de la función ConsultaPaginada() de la capa de lógica de negocio pasandole el módulo o sección, la tabla y los parámetros de la select..
Se modifica el ancho de la primera columna de la tabla y se obtiene el numero de registros de la tabla de la base de datos. Se activan y desactivan los botones de navegación de la tabla en base a la cantidad de registros por mostrar y se modifica el mensaje informativo situado en la parte inferior de la tabla.

```
private void CargarDatosTabla(string modulo1,
string tabla1, string campoRelacionado1, string
modulo2, string tabla2, string campoRelacionado2,
string parametrosSelect, int offset, int
registrosPorPagina, string criterioOrdenacion, int
posicionColumnaPK)
```

Función encargada de cargar en la tabla el resultado de una consulta paginada con join entre dos tablas .
Se establece como fuente de datos el resultado de la función ConsultaPaginada() de la capa de lógica de negocio pasandole los módulos o secciones, las tablas, los campos relacionados y los parámetros de la select..
Se modifica el ancho de la primera columna de la tabla y se obtiene el numero de registros de la tabla de la base de datos. Se activan y desactivan los botones de navegación de la tabla en base a la cantidad de registros por mostrar y se modifica el mensaje informativo situado en la parte inferior de la tabla.

MEMORIA TECNICA

```
private void CargarDatosTabla(string modulo1,
string tabla1, string campoRelacionado1, string
modulo2, string tabla2, string campoRelacionado2,
string modulo3, string tabla3, string
campoRelacionado3, string parametrosSelect,
int offset, int registrosPorPagina, string
criterioOrdenacion, int posicionColumnaPK)
```

Función encargada de cargar en la tabla el resultado de una consulta paginada con join entre tres tablas .
Se establece como fuente de datos el resultado de la función ConsultaPaginada() de la capa de lógica de negocio pasandole los módulos o secciones, las tablas, los campos relacionados y los parámetros de la select..
Se modifica el ancho de la primera columna de la tabla y se obtiene el numero de registros de la tabla de la base de datos. Se activan y desactivan los botones de navegación de la tabla en base a la cantidad de registros por mostrar y se modifica el mensaje informativo situado en la parte inferior de la tabla.

```
private void PersonalizarTabla(int redPrincipal,
int greenPrincipal, int bluePrincipal, int
redSecundario, int greenSecundario, int
blueSecundario)
```

Función encargada de modificar los colores de la tabla.

```
private void datagrid_DoubleClick(object sender,
EventArgs e)
```

Función encargada de llamar a la funcion btnModificar_Click(object sender, EventArgs e).

```
private void
comboBoxRegistrosPagina_SelectedIndexChanged(objec
t sender, EventArgs e)
```

Función encargada de modificar la variable registrosPorPagina y dependiendo del submenu seleccionado llamara a una sobrecarga o a otra de la función CargarDatosTabla()

METODOS FORMALMACENES.CS

A continuación se realizara una breve descripción de la utilidad de los métodos referentes al formulario de almacenes, así como de su funcionamiento interno.

```
private void btnClose_Click(object sender,
EventArgs e)
```

Función encargada de cerrar el formulario.

```
private void btnCancelar_Click(object sender,
EventArgs e)
```

Función encargada de cerrar el formulario.

```
private void FormAlmacenes_Load(object sender,
EventArgs e)
```

Función encargada de preparar el visionado del formulario dependiendo del modo en que se haya abierto el mismo.
Se establecen las fuentes de datos de los controles y se comprueba el modo de apertura. Si el formulario a sido abierto en modo editar o eliminar debera cargar los datos del elemento seleccionado en el formulario anterior.

```
private void btnAceptar_Click(object sender,
EventArgs e)
```

Función encargada de realizar alguna de las operaciones "CRUD" dependiendo del modo en el que se haya abierto el formulario.
Se comprueba el modo de apertura, así como que la validación del formulario sea correcta. Si esta condición se cumple, se creara el objeto pertinente y se llamara a la función de la BLL para insertar dicho objeto en la base de datos.

```
public bool ValidarAlmacen()
```

Función encargada de validar que los datos introducidos por el usuario en los controles del formulario sean correctos.

METODOS FORMEMPRESA.CS

A continuación se realizara una breve descripción de la utilidad de los métodos referentes al formulario de empresa, así como de su funcionamiento interno.

```
private void btnClose_Click(object sender,
EventArgs e)
```

Función encargada de cerrar el formulario.

```
private void btnCancelar_Click(object sender,
EventArgs e)
```

Función encargada de cerrar el formulario.

```
private void btnAceptar_Click(object sender,
EventArgs e)
```

Función encargada de realizar alguna de las operaciones "CRUD" dependiendo del modo en el que se haya abierto el formulario.
Se comprueba el modo de apertura, así como que la validación del formulario sea correcta. Si esta condición se cumple, se creara el objeto pertinente y se llamara a la función de la BLL para insertar dicho objeto en la base de datos.

```
private void btnNuevo_Click(object sender,
EventArgs e)
```

Función encargada de llamar a la funcion SetEnabledFormulario(true), la cual habilitara el formulario.

```
private void btnModificar_Click(object sender,
EventArgs e)
```

Función encargada de establecer el formulario en modo edicion con los valores de la empresa.

```
private void btnBuscarLogo(object sender,
EventArgs e)
```

Función encargada de facilitar al usuario la selección de una imagen alojada en su ordenador, para establecerla como logo de empresa.
Hace uso de la clase OpenFileDialog() para abrir una ventana de selección de archivos. Una vez haya seleccionado un archivo, muestro el nombre y la ruta en los controles pertinentes y cargo la imagen en el picturebox.

```
private bool IsModoCrear()
```

Función encargada de comprobar si nos encontramos en modo crear, para ello comprobamos la propiedad enabled de los botones btnNuevo y btnModificar.

```
private bool IsModoEditar()
```

Función encargada de comprobar si nos encontramos en modo editar, para ello comprobamos la propiedad enabled de los botones btnNuevo y btnModificar.

```
private void SetModoCrear()
```

Función encargada de habilitar el boton bntNuevo y deshabilitar el boton btnModificar.

```
private void SetModoEditar()
```

Función encargada de habilitar el botón btnModificary deshabilitar el botón btnNuevo.

```
private bool LimpiarFormConfiguracionEmpresa()
```

Función encargada de borrar las cajas de texto y devolver el formulario al estado inicial de su creación.

```
private bool ValidarEmpresa()
```

Funcion encargada de comprobar que el resto de funciones interiores que validan cada uno de los campos, devuelva true.

```
private bool ValidarRazonSocial()
```

Función encargada de comprobar si el campo es correcto, realizando diversas comprobaciones. Si no es correcto se mostrara un aviso y se cambiara el color del borde del control a rojo a modo de advertencia de su error.

```
private void txtTazonSocial_OnValueChanged(object
sender, EventArgs e)
```

Función encargada de llamar a la funcion ValidarRazonSocial() cada vez que se modifica el valor del input.

```
private bool ValidarNombreComercial()
```

Función encargada de comprobar si el campo es correcto, realizando diversas comprobaciones. Si no es correcto se mostrara un aviso y se cambiara el color del borde del control a rojo a modo de advertencia de su error.

```
private void
txtNombreComercial_OnValueChanged(object sender,
EventArgs e)
```

Función encargada de llamar a la funcion ValidarNombreComercial() cada vez que se modifica el valor del input.

```
private bool ValidarCIF()
```

Función encargada de comprobar si el campo es correcto, realizando diversas comprobaciones. Si no es correcto se mostrara un aviso y se cambiara el color del borde del control a rojo a modo de advertencia de su error.

```
private void txtCif_OnValueChanged(object sender,
EventArgs e)
```

Función encargada de llamar a la funcion ValidarCIF() cada vez que se modifica el valor del input.

```
private bool ValidarCodigoPostal()
```

Función encargada de comprobar si el campo es correcto, realizando diversas comprobaciones. Si no es correcto se mostrara un aviso y se cambiara el color del borde del control a rojo a modo de advertencia de su error.

```
private void txtCodigoPostal_OnValueChanged(object
sender, EventArgs e)
```

Función encargada de llamar a la funcion ValidarCodigoPostal() cada vez que se modifica el valor del input.

```
private bool ValidarDireccionFiscal()
```

Función encargada de comprobar si el campo es correcto, realizando diversas comprobaciones. Si no es correcto se mostrara un aviso y se cambiara el color del borde del control a rojo a modo de advertencia de su error.

```
private void
txtDireccionFiscal_OnValueChanged(object sender,
EventArgs e)
```

Función encargada de llamar a la funcion ValidarDireccionFiscal() cada vez que se modifica el valor del input.

```
private bool ValidarTelefono()
```

Función encargada de comprobar si el campo es correcto, realizando diversas comprobaciones. Si no es correcto se mostrara un aviso y se cambiara el color del borde del control a rojo a modo de advertencia de su error.

```
private void txtTelefono_OnValueChanged(object
sender, EventArgs e)
```

Función encargada de llamar a la funcion ValidarTelefono() cada vez que se modifica el valor del input.

```
private bool ValidarFax()
```

Función encargada de comprobar si el campo es correcto, realizando diversas comprobaciones. Si no es correcto se mostrara un aviso y se cambiara el color del borde del control a rojo a modo de advertencia de su error.

```
private void txtFax_OnValueChanged(object sender,
EventArgs e)
```

Función encargada de llamar a la funcion ValidarFax() cada vez que se modifica el valor del input.

```
private bool ExisteEmpresa()
```

Función encargada de comprobar si ya se ha creado una empresa con anterioridad. Para ello accede a la capa DAL y realiza una consulta.

```
private byte[] ImagenToByteArray(string
urlArchivo)
```

Función encargada de convertir una imagen en un array de bytes para su posterior guardado en la base de datos.
Se carga la imagen desde el path, Se vuelca la imagen sobre un memory stream y se convierte en array de bytes.

```
private Image ByteArrayToImagen(byte[] logoBytes)
```

Función encargada de convertir un array de bytes en una imagen para su posterior carga sobre el formulario.

MEMORIA TECNICA

METODOS FORMFAMILIAS.CS

A continuación se realizara una breve descripción de la utilidad de los métodos referentes al formulario de familias, así como de su funcionamiento interno.

```
private void btnClose_Click(object sender,
EventArgs e)
```

Función encargada de cerrar el formulario.

```
private void btnCancelar_Click(object sender,
EventArgs e)
```

Función encargada de cerrar el formulario.

```
private void FormFamilias_Load(object sender,
EventArgs e)
```

Función encargada de preparar el visionado del formulario dependiendo del modo en que se haya abierto el mismo.
Se establecen las fuentes de datos de los controles y se comprueba el modo de apertura. Si el formulario a sido abierto en modo editar o eliminar debera cargar los datos del elemento seleccionado en el formulario anterior.

```
private void btnAceptar_Click(object sender,
EventArgs e)
```

Función encargada de realizar alguna de las operaciones "CRUD" dependiendo del modo en el que se haya abierto el formulario.
Se comprueba el modo de apertura, así como que la validación del formulario sea correcta. Si esta condición se cumple, se creara el objeto pertinente y se llamara a la función de la BLL para insertar dicho objeto en la base de datos.

```
public bool ValidarFamilia()
```

Función encargada de validar que los datos introducidos por el usuario en los controles del formulario sean correctos.
Actualmente devuelve true siempre.

MEMORIA TECNICA

METODOS FORMMARCAS.CS

A continuación se realizara una breve descripción de la utilidad de los métodos referentes al formulario de marcas, así como de su funcionamiento interno.

```
private void btnClose_Click(object sender,
EventArgs e)
```

Función encargada de cerrar el formulario.

```
private void btnCancelar_Click(object sender,
EventArgs e)
```

Función encargada de cerrar el formulario.

```
private void FormMarcas_Load(object sender,
EventArgs e)
```

Función encargada de preparar el visionado del formulario dependiendo del modo en que se haya abierto el mismo.
Se establecen las fuentes de datos de los controles y se comprueba el modo de apertura. Si el formulario a sido abierto en modo editar o eliminar debera cargar los datos del elemento seleccionado en el formulario anterior.

```
private void btnAceptar_Click(object sender,
EventArgs e)
```

Función encargada de realizar alguna de las operaciones "CRUD" dependiendo del modo en el que se haya abierto el formulario.
Se comprueba el modo de apertura, así como que la validación del formulario sea correcta. Si esta condición se cumple, se creara el objeto pertinente y se llamara a la función de la BLL para insertar dicho objeto en la base de datos.

```
public bool ValidarMarca()
```

Función encargada de validar que los datos introducidos por el usuario en los controles del formulario sean correctos.
Actualmente devuelve true siempre.

METODOS FORMPRODUCTOS.CS

A continuación se realizara una breve descripción de la utilidad de los métodos referentes al formulario de productos, así como de su funcionamiento interno.

```
private void btnClose_Click(object sender,
EventArgs e)
```

Función encargada de cerrar el formulario.

```
private void btnCancelar_Click(object sender,
EventArgs e)
```

Función encargada de cerrar el formulario.

```
private void FormProductos_Load(object sender,
EventArgs e)
```

Función encargada de preparar el visionado del formulario dependiendo del modo en que se haya abierto el mismo.
Se establecen las fuentes de datos de los controles y se comprueba el modo de apertura. Si el formulario a sido abierto en modo editar o eliminar debera cargar los datos del elemento seleccionado en el formulario anterior.

```
private void btnAceptar_Click(object sender,
EventArgs e)
```

Función encargada de realizar alguna de las operaciones "CRUD" dependiendo del modo en el que se haya abierto el formulario.
Se comprueba el modo de apertura, así como que la validación del formulario sea correcta. Si esta condición se cumple, se creara el objeto pertinente y se llamara a la función de la BLL para insertar dicho objeto en la base de datos.

```
public bool ValidarProducto()
```

Función encargada de validar que los datos introducidos por el usuario en los controles del formulario sean correctos.
Actualmente devuelve true siempre.

```
private void txtIVA_OnValueChanged(object sender,
EventArgs e)
```

Función encargada de mostrar el valor de precio de venta al publico teniendo en cuenta los campos txtPrecioSinIva y txtIVA.
Si el usuario introduce precio sin iva pero no introduce iva, el pvp sera igual al precio sin iva.
Si el usuario no introduce precio sin iva, aunque introduzca iva, el pvp sera 0.
Si el usuario introduce ambos valores se realizara el calculo del pvp.

```
private void txtPrecioSinIva_OnValueChanged(object
sender, EventArgs e)
```

Función encargada de mostrar el valor de precio de venta al publico teniendo en cuenta los campos txtPrecioSinIva y txtIVA.
Si el usuario introduce precio sin iva pero no introduce iva, el pvp sera igual al precio sin iva.
Si el usuario no introduce precio sin iva, aunque introduzca iva, el pvp sera 0.
Si el usuario introduce ambos valores se realizara el calculo del pvp.

```
private void btnBuscarLogo(object sender,
EventArgs e)
```

Función encargada de facilitar al usuario la selección de una imagen alojada en su ordenador, para establecerla como logo de empresa.
Hace uso de la clase OpenFileDialog() para abrir una ventana de selección de archivos. Una vez haya seleccionado un archivo, muestro el nombre y la ruta en los controles pertinentes y cargo la imagen en el picturebox.

```
private byte[] ImagenToByteArray(string
urlArchivo)
```

Función encargada de convertir una imagen en un array de bytes para su posterior guardado en la base de datos.
Se carga la imagen desde el path, Se vuelca la imagen sobre un memory stream y se convierte en array de bytes.

```
private Image ByteArrayToImagen(byte[] logoBytes)
```

Función encargada de convertir un array de bytes en una imagen para su posterior carga sobre el formulario.