Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo e implementación del software de gestión para una PYME del sector de la automoción

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Jordi Benimeli Alemany

Tutor: Carlos David Martínez Hinarejos

2015/2016

Resumen

El presente Trabajo de Fin de Grado del Grado en Ingeniería Informática consiste en el desarrollo de una aplicación de escritorio para la gestión de una PYME del sector de la mecánica de vehículos. Dicho taller sigue funcionando en un programa desarrollado en MS-DOS, por lo que necesita una renovación de su software en todos los equipos.

El software actual gestiona los clientes, inventario, facturas, vehículos, ordenes de trabajo y empleados de la empresa. El nuevo software cubrirá los mismos procesos.

En la implementación del nuevo software se va a utilizar Visual Studio 2015 y SQL Server 2014.

Palabras clave: software de gestión, visual studio, C#, sql server.

Abstract

This Final Project of the Degree in Computer Engineering involves the developmnet of a desktop application for managing a SME in car mechanics sector. Currently, they continue using an application developed in MS-DOS, so it is necessary a renewal of the software.

The current software manages clients, inventory, invoices, cars, work orders and employees of the Company. The new software will cover the same processes.

In the implementation of the new software will be used Visual Studio 2015 and SQL Server 2014.

Keywords : management software, visual studio, C#, sql server.

Tabla de contenidos

1. Introducción

En los tiempos que corren, las tecnologías de la información y la comunicación viven en una constante evolución. Cada vez se depende más de la información digital y se deja a un lado la analógica. Esto implica que vivimos en una constante actualización, en la que, si no estás al día, te acabas estancando.

Gran parte de las PYMEs utilizan la fórmula de "si algo funciona, no lo toques", de forma que empezaron su andadura hace muchos años y siguen con la misma manera de trabajar de antaño. Esto no quiere decir que la empresa vaya mal, todo lo contrario, es sinónimo que de funciona. Pero eso no implica que no sea positiva una actualización. Este es el problema de la empresa en la que he desarrollado mi Trabajo de Fin de Grado.

1.1 Objetivo del Proyecto

El objetivo de este trabajo es aplicar una modernización necesaria en una empresa familiar que sigue utilizando el mismo software desde sus inicios. Para ello, se creará una aplicación de escritorio moderna, que pueda funcionar con los equipos informáticos presentes en la actualidad y que sea capaz de recibir actualizaciones periódicas en caso de necesitar nuevas funcionalidades.

1.2 Motivación

Haber elegido como Trabajo de Fin de Grado este proyecto tiene como razón principal las necesidades de la empresa que se ha implementado. Benimeli Motor S.L es una empresa regentada por mi familia en la que la gran mayoría de trabajadores son parientes míos y donde yo he trabajado muchos veranos.

En la empresa, se sigue utilizando un software MS-DOS desarrollado en el año 1990, que, dadas sus limitaciones, dificulta su escalabilidad y hace que cada vez sea más difícil adaptarse a los nuevos mercados. Desde que entré en el Grado de Ingeniería Informática, me pidieron que desarrollara el nuevo software de gestión para el negocio, pero evidentemente no me sentía capaz de realizarlo.

Ahora que he adquirido los conocimientos necesarios y después de haber hecho prácticas externas como desarrollador, me siento capaz de realizar aquello que llevan años pidiéndome y que la empresa de mi familia necesita para mejorar.

Por ello, la principal motivación es crear algo que realmente se necesita y que va a tener una utilidad durante muchos años.

1.3 Herramientas utilizadas

Siendo conscientes de que durante el Grado en Ingeniería Informática el lenguaje de programación más utilizado es Java, puede parecer extraño no utilizarlo para el proyecto. Aquí entran en juego mis prácticas externas.

Durante 6 meses he estado trabajando como desarrollador de C# en una empresa de ingeniería. Con esto me he visto capacitado para realizar el proyecto en este lenguaje de programación que, al no ser visto durante el grado, supone un atractivo extra.

Para ello, he utilizado Visual Studio 2015 y SQL Server 2014. En el apartado gráfico, el propio entorno de desarrollo nos proporciona la interfaz de programación gráfica Windows Forms, que se incluye como parte del .NET Framework.

Por su parte, la base de datos se ha escrito en Transact-SQL, el lenguaje utilizado en SQL Server. Es una extensión para SQL de Microsoft que incluye programación procedural, variables locales, varias funciones de soporte para procesamiento de *strings*, cambios en las sentencias DELETE y UPDATE y varias características más. Transact-SQL puede ser utilizado en otros lenguajes de programación como C o Java, aparte de los incluidos en el framework .NET.

En la fase de análisis se ha utilizado la herramienta ArgoUML para la creación de el diagrama de casos de uso y el diagrama de clases. Visual Studio ofrece una utilidad para la creación de estos diagramas, pero no está disponible en la versión gratuita para estudiantes. Por tanto, he usado esta alternativa gratuita igualmente válida.

Por último, para la creación de informes para las facturas y órdenes de trabajo, se utiliza la herramienta Crystal Reports.

2. Descripción del sistema

2.1 Usuarios

Hay dos tipos de usuarios, los administradores y los técnicos.

Los administradores tendrán acceso a la totalidad de la aplicación y serán los únicos capaces de realizar tareas de gestión como crear e imprimir facturas, agregar productos al inventario, insertar nuevos clientes y listar clientes u órdenes de trabajo.

Los técnicos, por su parte, podrán realizar tareas como crear órdenes de trabajo, agregar vehículos a los clientes y consultar facturas o vehículos existentes.

2.2 Requisitos funcionales

Los requisitos funcionales definen funciones o componentes del sistema. Para definir un requisito funcional, se debe describir un conjunto de entradas, comportamientos y salidas generado. A continuación, se listan los requisitos funcionales y se describen según este esquema.

Login de usuario

- **Entradas:** Es necesario introducir usuario y contraseña para acceder a la aplicación.
- **Comportamientos:** Se realiza una consulta a la base de datos para comprobar que el usuario y contraseña sean correctos.
- **Salidas:** Si usuario y contraseña son correctos, se accede a la aplicación. En caso contrario, aparece un mensaje de login incorrecto.

Añadir usuario (Administrador)

- **Entradas:** Se rellena un formulario para dar de alta un nuevo usuario.
- **Comportamientos:** Se comprueba que no haya campos vacíos en el formulario y que el nombre de usuario no exista en la base de datos.
- **Salidas:** Si las comprobaciones son correctas, aparece un mensaje de usuario añadido. En caso contrario, se informa del campo a rellenar/modificar.

Modificar usuario (Administrador)

- **Entradas:** Se modifican los campos necesarios en el formulario de datos del usuario.
- **Comportamientos:** Se comprueba que se hayan producido cambios y que los campos necesarios no estén vacíos.
- **Salidas:** En caso de que los valores sean correctos, se modifica el usuario en la base de datos. En caso contrario, se informa de la ausencia de modificaciones o del campo a rellenar.

Eliminar usuario (Administrador)

- **Entradas:** Se introduce el usuario a eliminar.
- **Comportamientos:** Se muestra un mensaje de confirmación para la eliminación.
- **Salidas:** En caso de confirmar la operación, se elimina el usuario de la base de datos.

Introducir cliente (Administrador)

- **Entradas:** Se rellena un formulario con los datos del cliente.

- **Comportamientos:** Se comprueba que los campos necesarios no estén vacíos.
- **Salidas:** En caso de que los valores sean correctos, se introduce el cliente en la base de datos. En caso contrario, se informa del campo (o los campos) a rellenar.

Modificar cliente (Administrador)

- **Entradas:** Se modifican los campos necesarios en el formulario de datos del cliente.
- **Comportamientos:** Se comprueba que se hayan producido cambios y que los campos necesarios no estén vacíos.
- **Salidas:** En caso de que los valores sean correctos, se modifica el cliente en la base de datos. En caso contrario, se informa de la ausencia de modificaciones o del campo a rellenar.

Eliminar cliente (Administrador)

- **Entradas:** Se introduce el cliente a eliminar.
- **Comportamientos:** Se muestra un mensaje de confirmación para la eliminación.
- **Salidas:** En caso de confirmar la operación, se elimina el cliente de la base de datos.

Introducir producto al inventario (Administrador)

- **Entradas:** Se rellena un formulario con los datos del producto.
- **Comportamientos:** Se comprueba que los campos necesarios no estén vacíos y que la referencia de producto no exista en la base de datos.
- **Salidas:** En caso de que los valores sean correctos, se introduce el producto en la base de datos. En caso contrario, se informa del campo a rellenar/modificar.

Modificar producto (Administrador)

- **Entradas:** Se modifican los campos necesarios en el formulario de datos del producto.
- **Comportamientos:** Se comprueba que se hayan producido cambios y que los campos necesarios no estén vacíos.
- **Salidas:** En caso de que los valores sean correctos, se modifica el producto en la base de datos. En caso contrario, se informa de la ausencia de modificaciones o del campo a rellenar.

Eliminar producto (Administrador)

- **Entradas:** Se introduce el producto a eliminar.
- **Comportamientos:** Se muestra un mensaje de confirmación para la eliminación.
- **Salidas:** En caso de confirmar la operación, se elimina el producto de la base de datos.

Crear factura (Administrador)

- **Entradas:** Se rellena un formulario con los datos de la factura
- **Comportamientos:** Se comprueba que los campos necesarios no estén vacíos, que tenga asociada una orden de trabajo y que el número de factura no exista en la base de datos.
- **Salidas:** En caso de que los valores sean correctos, se guarda la factura en la base de datos y se pregunta al usuario si quiere imprimirla. En caso contrario, se informa del campo a rellenar/modificar.

Imprimir factura (Administrador)

- **Entradas:** Se introduce el número de factura a imprimir
- **Comportamientos:** Se comprueba que la factura exista en la base de datos.
- **Salidas:** En caso de que exista, aparece un mensaje de confirmación y se imprime la factura.

Listar clientes (Administrador)

- **Entradas:** Se introducen los filtros requeridos en el listado.
- **Comportamientos:** Se realiza una consulta a la base de datos con los filtros introducidos.
- **Salidas:** Se abre una ventana con el listado devuelto por el sistema.

Listar órdenes de trabajo (Administrador)

- **Entradas:** Se introducen los filtros requeridos en el listado.
- **Comportamientos:** Se realiza una consulta a la base de datos con los filtros introducidos.
- **Salidas:** Se abre una ventana con el listado devuelto por el sistema.

Agregar vehículo al cliente (Administrador y técnico)

- **Entradas:** Se rellena un formulario con los datos del vehículo.
- **Comportamientos:** Se comprueba que los campos necesarios no estén vacíos, que tenga asociado un cliente y que la matrícula introducida no exista en la base de datos.
- **Salidas:** En caso de que los valores sean correctos, se guarda el vehículo en la base de datos. En caso contrario, se informa del campo a rellenar/modificar.

Modificar vehículo (Administrador y técnico)

- **Entradas:** Se modifican los campos necesarios en el formulario de datos del vehículo.
- **Comportamientos:** Se comprueba que se hayan producido cambios y que los campos necesarios no estén vacíos.
- **Salidas:** En caso de que los valores sean correctos, se modifica el vehículo en la base de datos. En caso contrario, se informa de la ausencia de modificaciones o del campo a rellenar.

Eliminar vehículo (Administrador y técnico)

- **Entradas:** Se introduce el vehículo a eliminar.
- **Comportamientos:** Se muestra un mensaje de confirmación para la eliminación.
- **Salidas:** En caso de confirmar la operación, se elimina el vehículo de la base de datos.

Crear orden de trabajo (Administrador y técnico)

- **Entradas:** Se rellena un formulario con los datos de la orden de trabajo.
- **Comportamientos:** Se comprueba que los campos necesarios no estén vacíos, que tenga asociado un vehículo y que el número de orden de trabajo no exista en la base de datos.
- **Salidas:** En caso de que los valores sean correctos, se guarda la orden de trabajo en la base de datos y se imprime una copia para el técnico. En caso contrario, se informa del campo a rellenar/modificar.

Consultar factura (Administrador y técnico)

- **Entradas:** Se introduce el número de factura.

- **Comportamientos:** Se realiza una consulta a la base de datos con el número de factura.
- **Salidas:** Si el número de factura es correcto, se abre una ventana con los datos de la factura. En caso contrario, se muestra un mensaje de número de factura incorrecto.

Consultar cliente (Administrador y técnico)

- **Entradas:** Se introduce el nombre o el identificador de cliente.
- **Comportamientos:** Se realiza una consulta a la base de datos con los datos introducidos.
- **Salidas:** Si la base de datos devuelve algún cliente, se abre una ventana y se muestran sus datos. En caso contrario, se muestra un mensaje de cliente incorrecto.

2.3 Requisitos no funcionales

Los requisitos no funcionales especifican los criterios para juzgar el comportamiento del sistema entero. Estos requisitos no describen las funciones que realiza el sistema, sino las características de funcionamiento de todo el sistema. A continuación, se listan este tipo de requisitos para nuestra aplicación.

- **Rendimiento:** el rendimiento del sistema debe ser igual o superior que el sistema al que va a sustituir.
- **Seguridad:** el software dispone de un sistema de autenticación por usuario y contraseña, con lo que se restringe el acceso a la información. Por otro lado, la base de datos contiene información de los clientes, por lo que se debe asegurar que el acceso está protegido. Además, se deberán realizar copias de seguridad periódicas de la base de datos.
- **Facilidad de uso:** los usuarios deben experimentar una rápida adaptación al nuevo sistema, intentando que su modo de trabajar sea lo más parecido posible al del sistema antiguo.
- **Estabilidad:** se debe implementar el servidor en un buen equipo para asegurarse los menos fallos posibles en su funcionamiento.
- **Escalabilidad:** el sistema debe poderse instalar en cualquier equipo nuevo que se añada a la red interna sin que haya problemas en los accesos al servidor. Además, la base de datos debe poder ser ampliada cuando sea necesario más espacio.
- **Soporte:** el sistema debe de ser fácil de instalar y capaz de ser actualizado.

3. Análisis

Durante la fase de análisis se ha examinado la especificación de requisitos para realizar el diseño de la aplicación. La captura, análisis y especificación de requisitos es una parte crucial en el desarrollo del software; de esta etapa depende en gran medida el resultado final de la aplicación.

Para ello, una vez se han realizado las entrevistas necesarias con el cliente para la especificación de requisitos, se procede a realizar un estudio de los casos de usos de la aplicación.

3.1 Diagrama de casos de uso

Los diagramas de casos de uso son una herramienta esencial para obtener una mayor comprensión del sistema a desarrollar. Karl E. Wiegers describe los casos de uso como "escenarios en los cuales el usuario interactúa con el sistema que se está definiendo para lograr cierto objetivo específico o realizar alguna tarea en particular" (*Software Development Magazine*, mayo de 1997).

En los diagramas de casos de uso, podemos encontrar tres componentes esenciales:

1. Actores: representan a los usuarios que interaccionan con el sistema. Estos actores no representan a una persona en particular, sino que representan los roles que los diferentes usuarios pueden adquirir en el manejo de la aplicación. Por ejemplo, el actor Administrador no implica que exista una persona en la empresa que sea el Administrador, sino que ese rol se le puede asignar al jefe de la empresa, al personal de administración o a quien sea necesario.
2. Casos de uso: representan las operaciones o tareas específicas que realiza el sistema tras una orden externa, ya sea por petición de un actor o por invocación desde otro caso de uso.
3. Relaciones: hay tres tipos:
   - Asociación
     - Es el tipo de relación más básica. Indica la invocación de un caso de uso, ya sea desde un actor o desde otro caso de uso.
   - Dependencia
     - Puede ser de extensión `(<<extend>>)`, donde el caso de uso base incorpora el comportamiento de otro caso de uso y puede llegar a usarlo, o de inclusión `(<<include>>)`, donde el caso de uso base engloba la acción del otro.
   - Generalización
     - Indica que un caso de uso es un caso particular de uno más general (herencia).

Para la aplicación desarrollada, el diagrama de casos de uso es el que se presenta en la página 13.

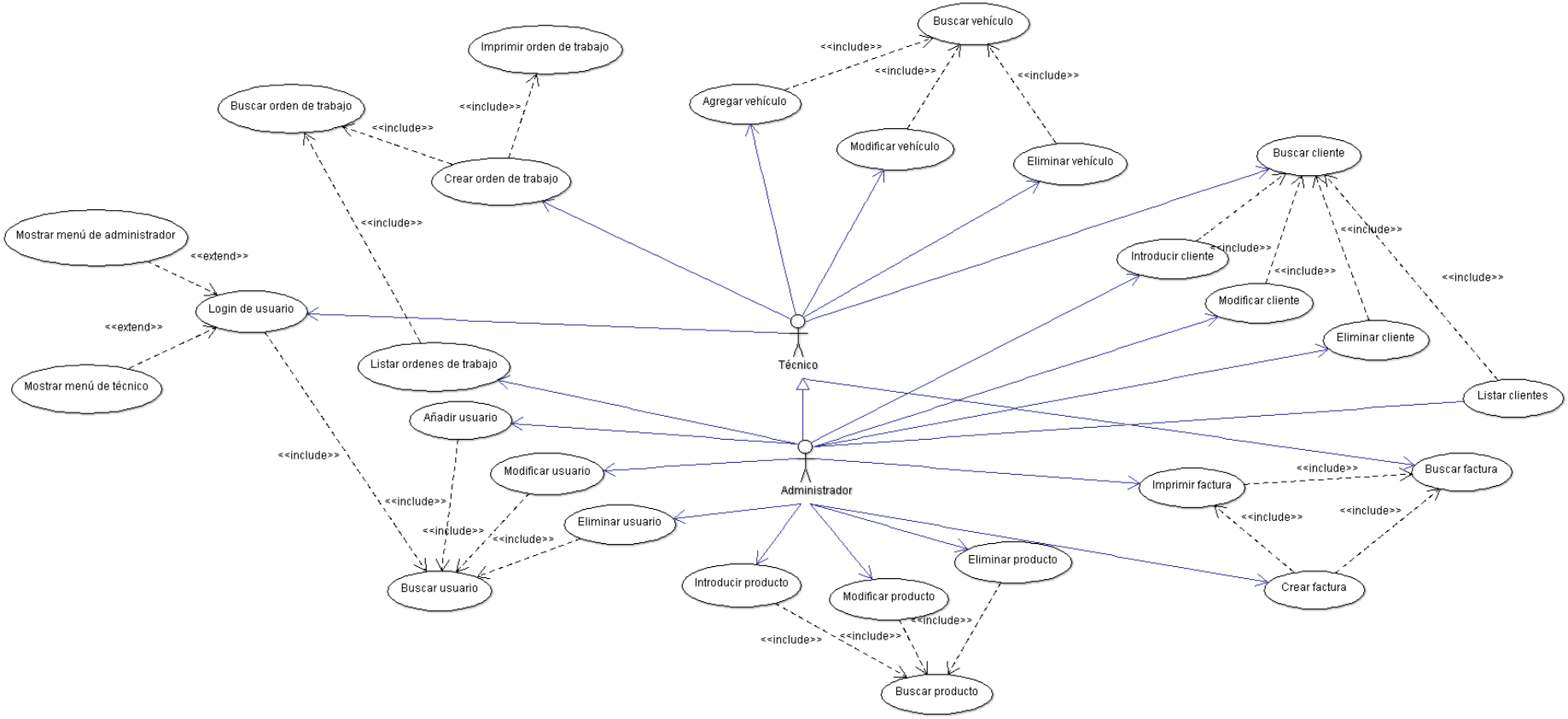


Ilustración 1 - Diagrama de casos de uso

3.2 Diagrama de clases

Los diagramas de clases son una herramienta para mostrar las entidades que participan en el desarrollo del sistema, mostrando cómo se relacionan entre ellas y qué atributos tiene cada entidad. Estos diagramas son una herramienta esencial en la programación orientada a objetos.

Una clase es la descripción de un grupo de objetos con estructura, comportamiento y relaciones similares. Están compuestas por:

- Nombre: nombre de la clase.
- Atributos: información del objeto.
- Métodos: acciones que puede realizar el objeto.

Al igual que en los casos de uso, las clases se pueden relacionar entre sí de diferentes formas:

- Herencia: 
  - Indica que una subclase hereda los métodos y atributos especificados por una superclase; por tanto, la subclase poseerá los métodos y atributos visibles de la superclase además de los suyos propios.
- Agregación: 
  - Inclusiva: es una relación estática donde el tiempo de vida del objeto incluido está condicionado por el tiempo de vida del que lo incluye. Si el objeto base se destruye, el objeto incluido también lo hará. Se representa pintando el rombo de color negro.
  - Referencial: es una relación dinámica donde los tiempos de vida de los objetos son independientes. Se representa pintado el rombo de color blanco.
- Asociación:
  - Es una relación estructural donde los objetos de un elemento están conectados con los objetos del otro.
  - Toda asociación es bidireccional, es decir, se puede navegar en los dos sentidos, desde objetos de una clase a objetos de la otra.
  - Las asociaciones tienen nombre para ayudar a comprender la relación entre las clases.
  - En los extremos de las relaciones, se indica la multiplicidad de cada objeto en la relación. Por ejemplo: una relación 1…1------0…* indica que cada objeto A se puede relacionar con 0 o muchos objetos B, pero cada objeto B debe tener relacionado 1 objeto A.

Para la aplicación desarrollada, el diagrama de clases empleado es el que se presenta en la página 15.

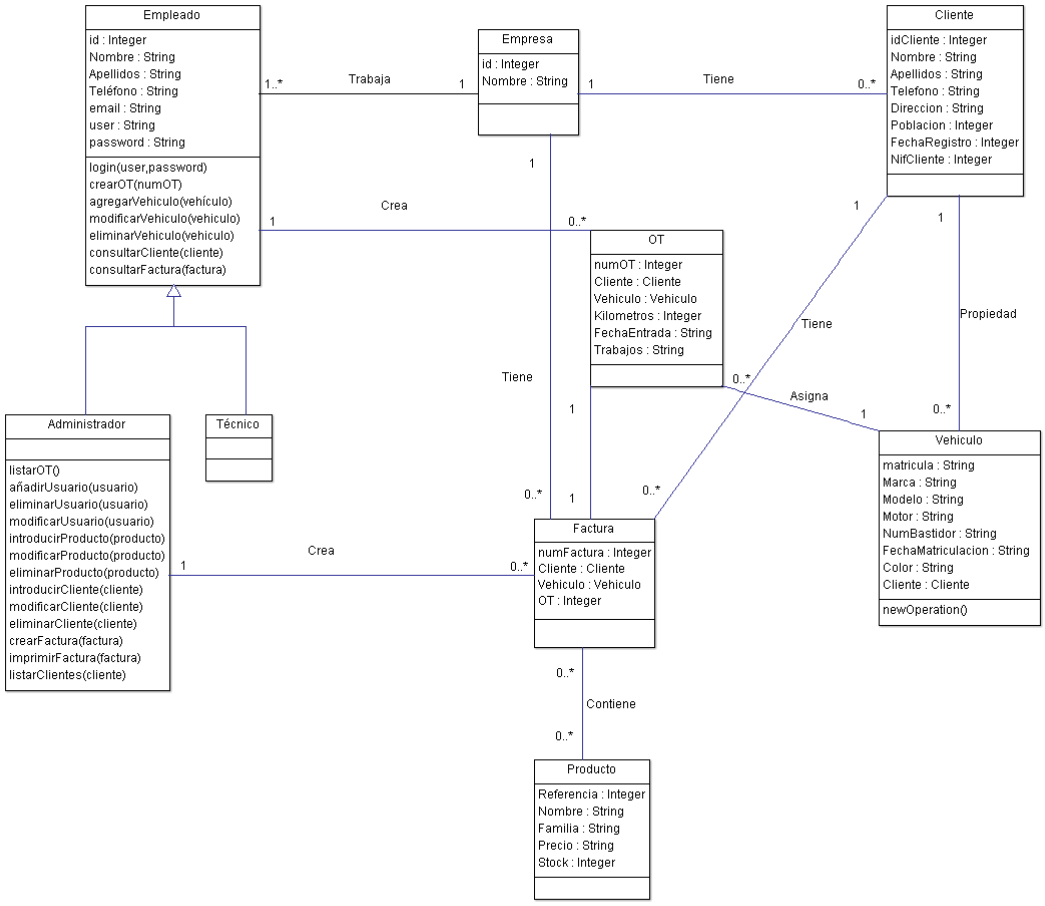


Ilustración 2 - Diagrama de clases

3.3 Mockups del software

Un *mockup* es un boceto utilizado para hacer pruebas y demostraciones de lo que en un futuro puede llegar a ser el software a desarrollar. Se utilizan para mostrar al cliente cómo será su aplicación y poder cambiar lo que este le pida antes de empezar el desarrollo.

El primer *mockup* de este proyecto corresponde a la pantalla de *login*. En ella el usuario deberá introducir su usuario y contraseña para acceder a la aplicación.

Ilustración 3 - Login

Una vez pasado con éxito el *login*, accedemos a la pantalla principal, donde encontramos los botones para acceder a todas las pantallas.

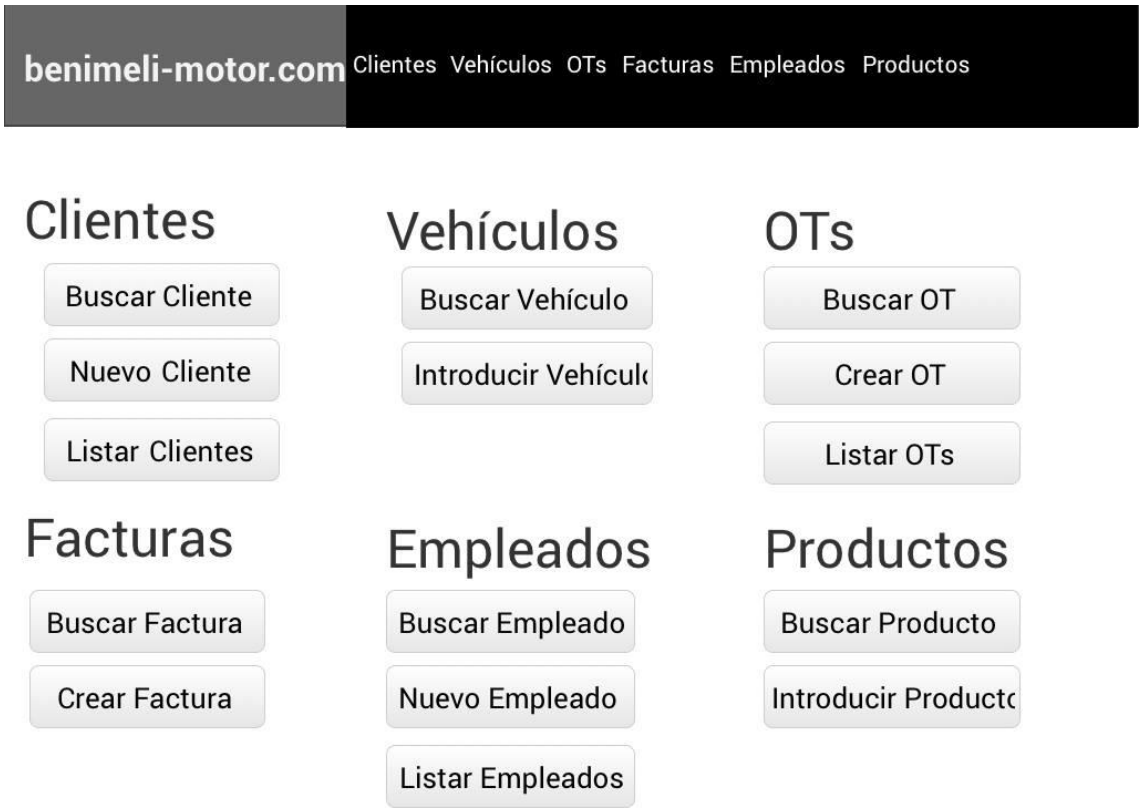


Ilustración 4 - Pantalla principal

Si pulsamos sobre buscar cliente, aparecerá un *textbox* para buscar el cliente en cuestión y un *grid* donde se mostrarán los resultados.

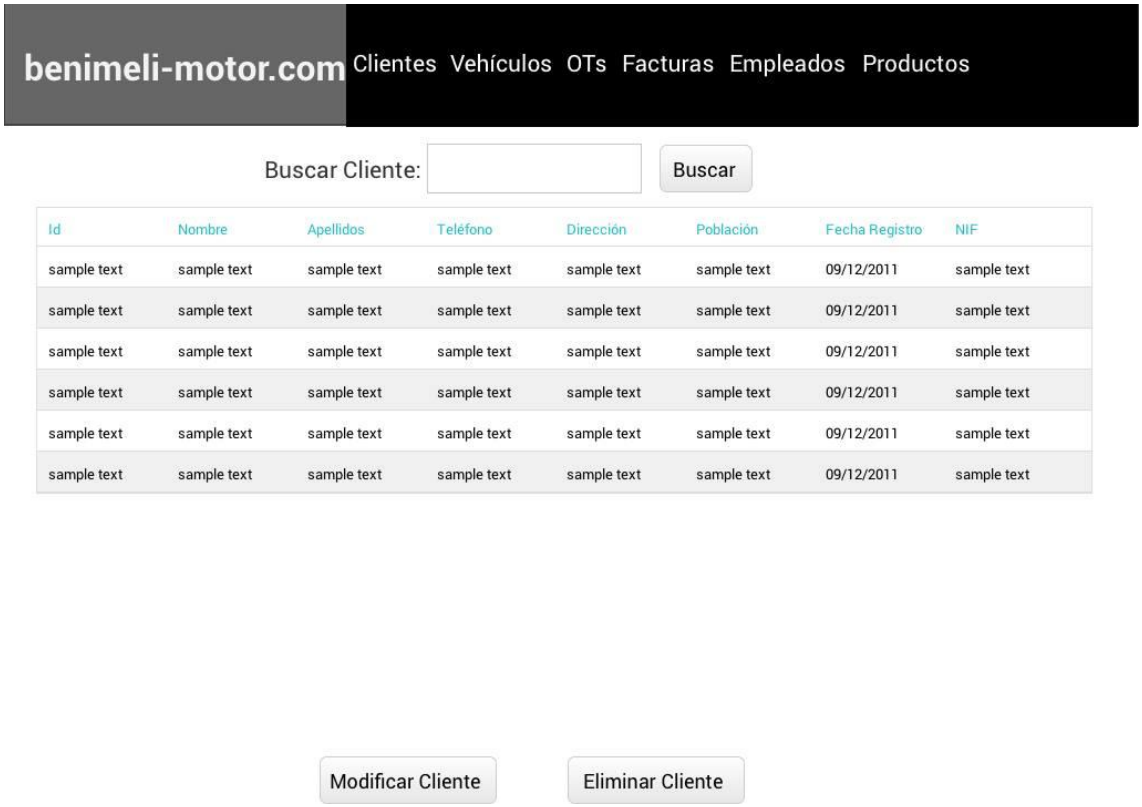


Ilustración 5 - Buscar cliente

El botón de nuevo cliente nos abre un formulario donde deberemos rellenar todos los datos del nuevo cliente. Tanto vehículos como productos tendrán una pantalla similar con sus respectivos campos.

Ilustración 6 - Insertar cliente

El botón de listar clientes nos abrirá un listado de todos los clientes desde donde podemos acceder a cada uno de ellos ya sea para modificarlo o eliminarlo.

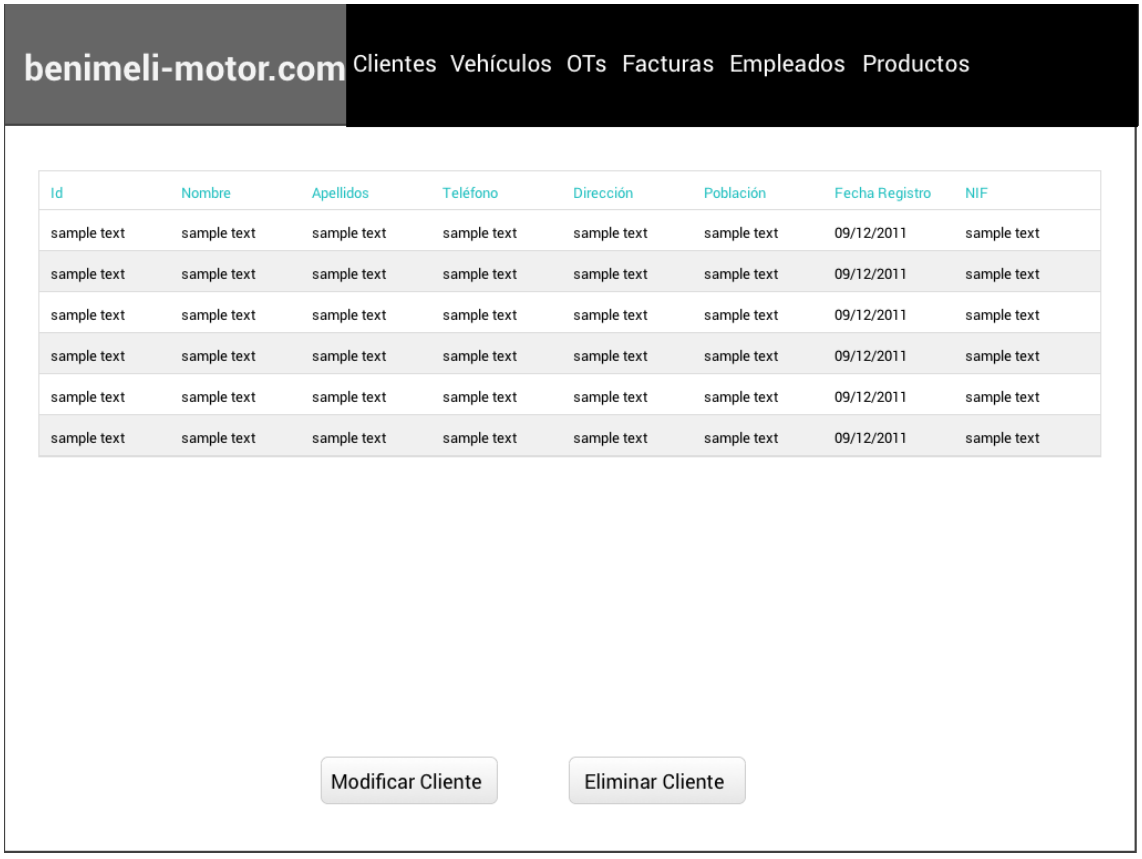


Ilustración 7 - Listar clientes

Crear OT (orden de trabajo) nos abre un formulario donde tendremos que introducir los campos necesarios y escribir en el *grid* los trabajos que se vayan a realizar en el vehículo.

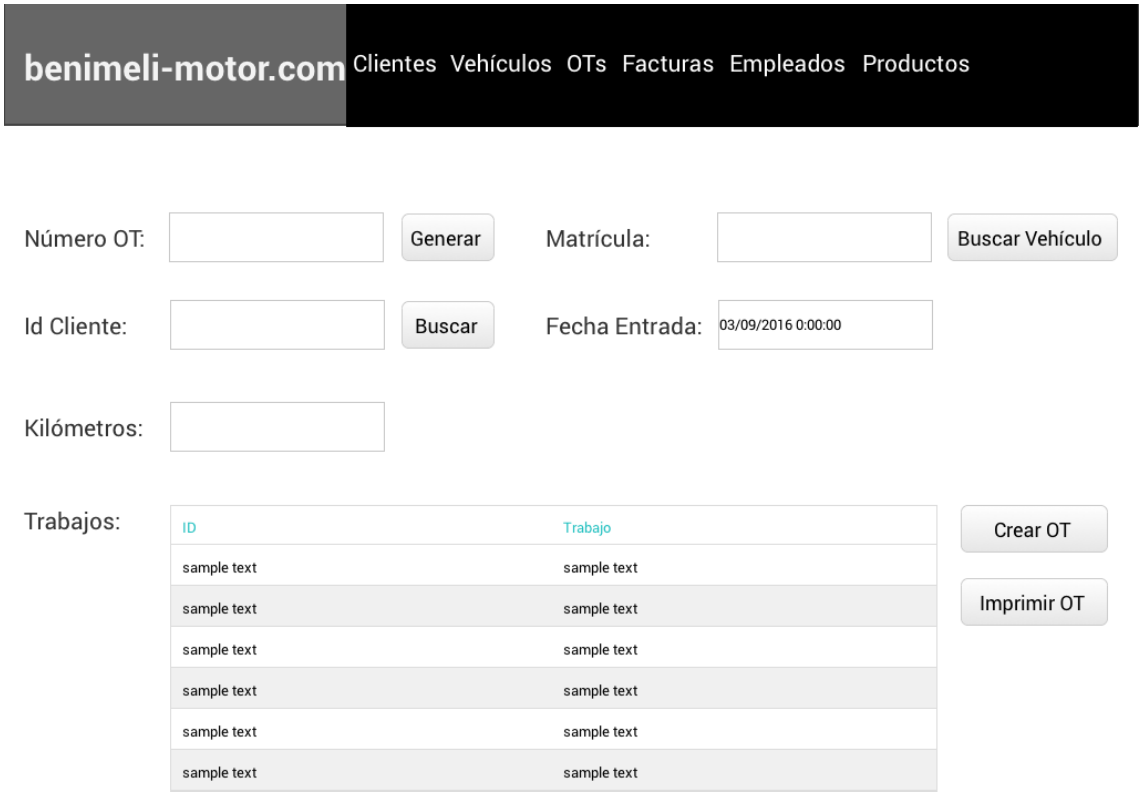


Ilustración 8 - Crear OT

En el formulario anterior no será necesario introducir manualmente la matrícula del vehículo ni el id del cliente, sino que desde el botón buscar de cada campo accedemos al siguiente formulario (o el equivalente en el caso de buscar un vehículo) que nos buscará el cliente o vehículo deseado.

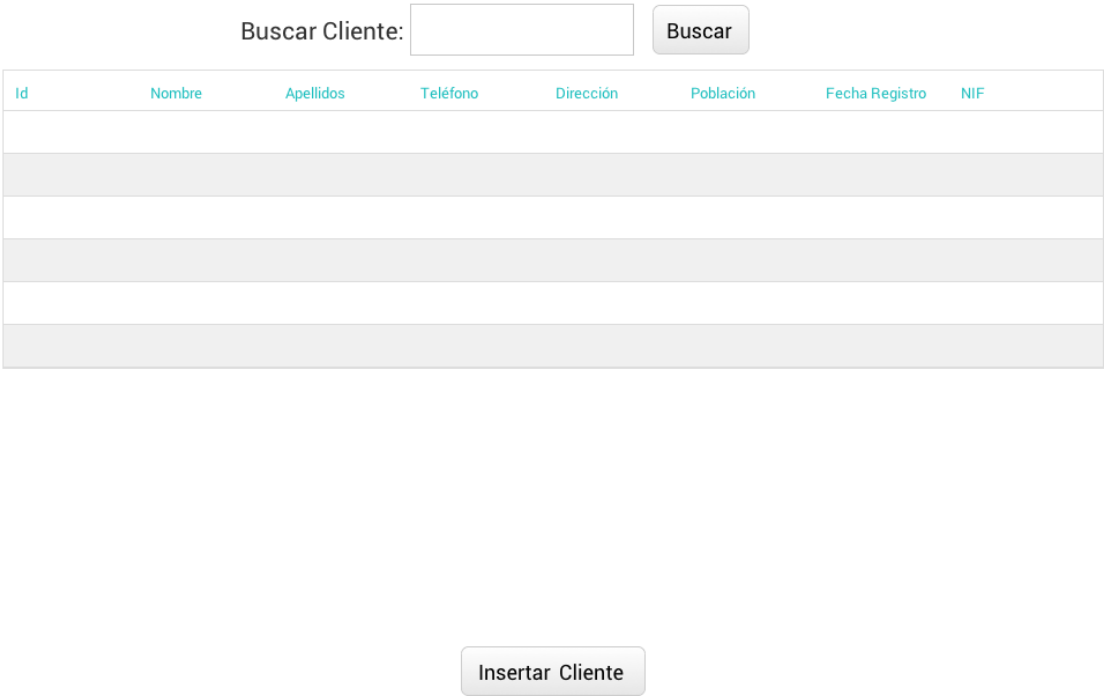


Ilustración 9 - Ventana de buscar cliente

En el caso de introducir un nuevo producto, tendremos que introducir la referencia deseada para el producto y acto seguido pulsar el botón de comprobar para confirmar que la referencia no exista.

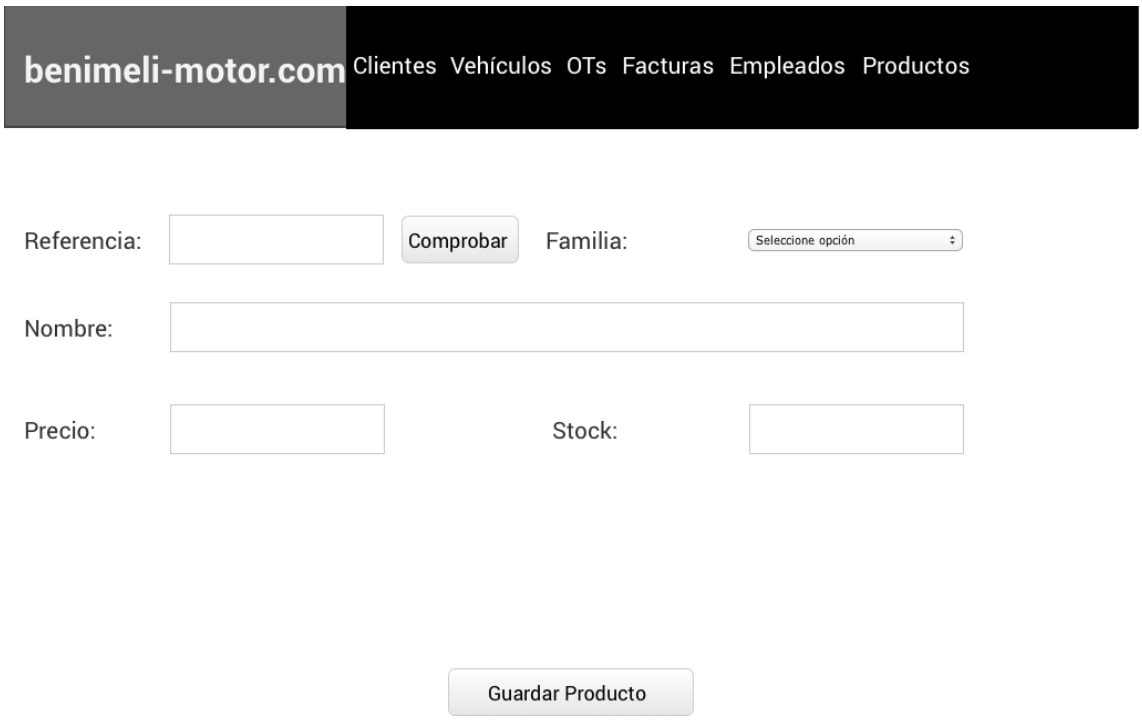


Ilustración 10 - Introducir producto

Si queremos insertar un nuevo empleado en el sistema, tendremos que rellenar el formulario en cuestión, donde se tendrá que asignar el usuario y la contraseña que se desee y escoger su nivel en el *comboBox*.

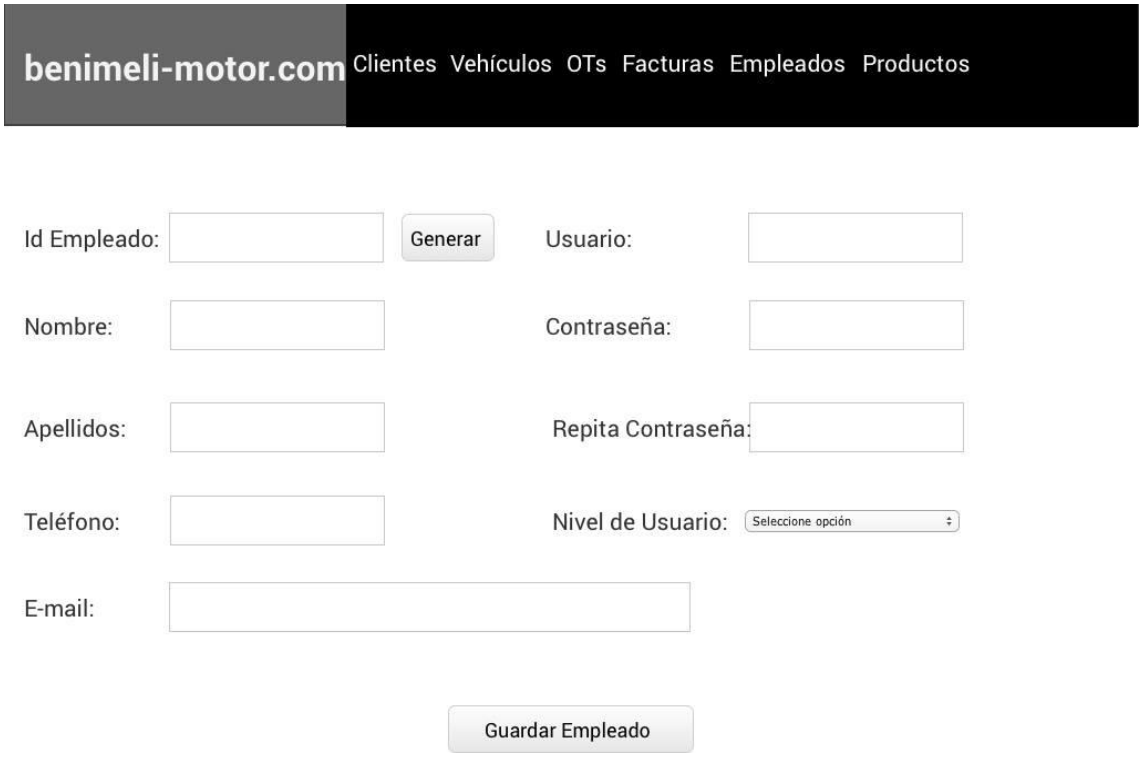


Ilustración 11 - Nuevo empleado

4. Diseño del software

El diseño del software es el primer paso de la fase de desarrollo de cualquier producto o sistema. Este comienza una vez que se han analizado y modelado los requerimientos del sistema y estudia la arquitectura y los patrones que se van a utilizar durante el desarrollo para alcanzar los requerimientos del sistema definidos anteriormente.

4.1 Arquitectura multicapa

La arquitectura multicapa (o programación por capas) es un estilo de programación que tiene por objetivo separar la lógica de negocios de la lógica de diseño.

Con esto conseguimos una independencia entre los distintos niveles, cosa que nos permite (en caso de ser necesario alguna modificación en el código) ir directamente al nivel requerido para realizar el cambio.

La programación por capas es una técnica de la ingeniería del software propia de la programación orientada a objetos y, aunque es posible diseñar cuantas capas sean necesarias, el diseño más utilizado actualmente (y que es utilizado en este software) es el diseño en tres capas: la capa de presentación, la capa de negocio y capa de datos.

Ilustración 12 - Gráfico de arquitectura multicapa

4.1.1 Capa de presentación

Es más conocida como interfaz de usuario. Es con lo que interactúa el usuario final y debe ofrecerle una presentación entendible y fácil de usar. Tiene que ser capaz de recoger la información proporcionada por el usuario y devolverle la información que éste último le exija.

La capa de presentación se comunica con la capa de negocio para ofrecer y obtener la información proporcionada por el usuario.

4.1.2 Capa de negocio

Es la capa encargada de realizar todas las peticiones del usuario y devolver la respuesta de estas peticiones. Aquí se establecen todas las reglas que se tendrán que cumplir en base a cada proceso solicitado por el usuario.

La capa de negocio (o capa "lógica" en este proyecto) es la única que se relaciona con dos capas. Por una parte, se relaciona con la capa de presentación cuando ésta le comunica las peticiones del usuario (y para devolverle las respuestas) y con la capa de datos para solicitar acceso a la base de datos (ya sea para introducir u obtener información).

4.1.3 Capa de datos

La capa de datos (o capa "persistencia" en este proyecto) es la única que tiene acceso a la base de datos del software y debe asegurar el acceso a la información de una forma controlada y segura.

Ésta capa es la encargada de proporcionar toda la información que le es solicitada desde la capa de negocio.

4.1.4 Ventajas e inconvenientes de la arquitectura multicapa

- Ventajas:
    - Reutilización de capas.
    - Facilita la estandarización.
    - Incremento de la seguridad gracias a las limitaciones de dependencias entre capas.
    - Facilidad de modificación gracias a la separación.
- Inconvenientes:
    - A veces no se logra realizar correctamente un cambio y requiere una cascada de cambios en varias capas.
    - Se produce una pérdida de eficiencia.
    - Trabajo innecesario o redundante entre varias capas.
    - Supone una mayor carga en la red debido al tráfico generado.

5. Implementación

Durante la fase de implementación en el desarrollo de un software, se procede a programar o implementar los diseños especificados durante la fase de diseño del software.

En la implementación se debe asegurar que el sistema funcione de acuerdo a los requerimientos del análisis y que permita a los usuarios operar de la forma que se estableció.

5.1 Tecnologías utilizadas

5.1.1 C#

El lenguaje de programación utilizado ha sido C#. Es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET. Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET, similar al de Java.

Los programas de C# se ejecutan en .NET Framework, un componente que forma parte de Windows y que incluye un sistema de ejecución virtual denominado Common Language Runtime (el equivalente al JRE de Java).

El .NET Framework se incluye en el sistema operativo Windows desde Windows Vista, además de poderse instalar posteriormente en Windows XP. Esto implica que cualquier programa de C# podrá ser ejecutado en estos sistemas operativos sin necesidad de instalar ningún componente extra (y siempre que el hardware del ordenador lo permita).

La última versión de .NET Framework es la 4.6.2, lanzada el 2 de agosto del 2016. Incluye todas las versiones del Framework desde la 4.0. Eso significa que para lanzar programas compilados en una versión anterior del Framework, será necesario instalar la versión 3.5 (que incluye todas las versiones desde la 2.0).

5.1.2 Transact-SQL

Transact-SQL es una extensión de SQL propiedad de Microsoft y Sybase. SQL es un lenguaje de consulta para sistemas de bases de datos relacionales. Con Transact-SQL extendemos este lenguaje para convertirlo en un lenguaje de programación.

Con esta extensión conseguimos añadir a SQL instrucciones y elementos tales como procedimientos almacenados, variables y tipos de datos propios, funciones, *triggers* y *scripts*.

5.2 Herramientas utilizadas

5.2.1 Visual Studio 2015

Microsot Visual Studio es un entorno de desarrollo integrado (IDE) para sistemas operativos Windows. Soporta multiples lenguajes de programación tales como C++, C#, Visual Basic .NET, F#, Java, Python, Ruby o PHP y entornos de desarrollo web como ASP.NET o Django.

Visual Studio incluye una interfaz de programación gráfica llamada Windows Forms que proporciona acceso a los elementos de la interfaz de Microsoft Windows. Este proyecto ha sido realizado utilizando esta interfaz.

La primera versión de este entorno de desarrollo fue Visual Studio 97 (lanzada en mayo del 1997) y fue el primer intento de Microsoft de usar el mismo entorno de desarrollo para múltiples

lenguajes. La última versión estable es Visual Studio 2015 Update 3 y fue lanzada el 27 de junio de 2016.

Visual Studio es un entorno de desarrollo de pago, pero desde la versión 2005, Microsoft ofrece gratuitamente las versiones Express del programa, que son versiones básicas separadas por lenguajes de programación para estudiantes y programación amateur.

5.2.2 SQL Server 2014

SQL Server es un sistema de gestión de base de datos relacionales propiedad de Microsoft. Solo está disponible para el sistema operativo Windows y es, junto con MySQL, el sistema más utilizado en Windows. Actualmente, existe una versión beta de la última versión del programa disponible para Linux y se espera que la versión final sea lanzada a mediados de 2017.

Utiliza el lenguaje de programación Transact-SQL y tiene, entre otras características, soporte de transacciones, soporte de procedimientos almacenados, un entorno gráfico de administración (SQL Server Management Studio), permite trabajar en modo cliente-servidor y permite administrar información de otros servidores de datos.

Su primera versión fue lanzada el año 1989 para el sistema operativo OS/2 y no fue hasta la versión 4.21 cuando cambió al sistema operativo Windows NT. La última versión estable es SQL Server 2016 y fue lanzada el 1 de junio de 2016.

5.2.2 GitHub

GitHub es una plataforma de desarrollo colaborativo utilizada para alojar proyectos utilizando el sistema de control de versiones Git. GitHub proporciona una total integración con Visual Studio, ofreciendo este último la posibilidad de agregar la extensión para utilizar GitHub durante su instalación.

El código se almacena de forma pública en la plataforma, pero existe la posibilidad de crear repositorios privados mediante pago. No obstante, GitHub se ha metido de lleno en el área de la educación y ofrece de forma totalmente gratuita repositorios privados durante todos los años que seas estudiante.

En el anexo 1, hay un enlace al repositorio de GitHub de este proyecto.

5.2.3 ArgoUML

ArgoUML es una aplicación de diagramado UML escrita en Java. Se ofrece de forma totalmente gratuita bajo la licencia BSD. Dado que es una aplicación Java, está disponible en cualquier plataforma soportada por Java.

Se ha utilizado para realizar el diagrama de casos de uso y el diagrama de clases del capítulo 3.

5.2.4 Justinmind Prototyper

Prototyper es una herramienta para la creación de bocetos o *Mockups*. Forma parte de la *startup* española Justinmind, que ha conseguido afincarse en Silicon Valley después de que grandes empresas como Google, Paypal o Sony utilicen su herramienta.

Se ofrece en versión PRO y en versión FREE, teniendo esta primera una versión de prueba de 30 días que posteriormente pasa a la versión gratuita.

5.3 Detalles del software

5.3.1 Login

Durante la programación del software, se ha implementado una ventana de login para separar los distintos casos de uso según el rol del usuario. La ventana de login solicita una consulta a la base de datos para comprobar las credenciales introducidas por el usuario.

Mediante las utilidades proporcionadas por Visual Studio, se ha incluido una encriptación SHA1 de las claves en la base de datos. De este modo, al registrar un nuevo usuario, se fusionan el usuario y la contraseña (antes de la encriptación) para aumentar la seguridad al paso por la función de encriptación.

La ventana de login presenta el siguiente aspecto:

Ilustración 13 - Login de la aplicación

5.3.2 Capturas de la aplicación

Después de ingresar correctamente a través del *login*, accederemos a la pantalla principal, donde a primera vista aparecen los botones que dan acceso a las diferentes pantallas de la aplicación.

Por otro lado, en la parte superior aparece una barra de menú desde la que podemos acceder a todas las pantallas y desde cualquiera de ellas. Además, haciendo *click* en el logo de la aplicación desde cualquier pantalla, se vuelve a esta pantalla principal.

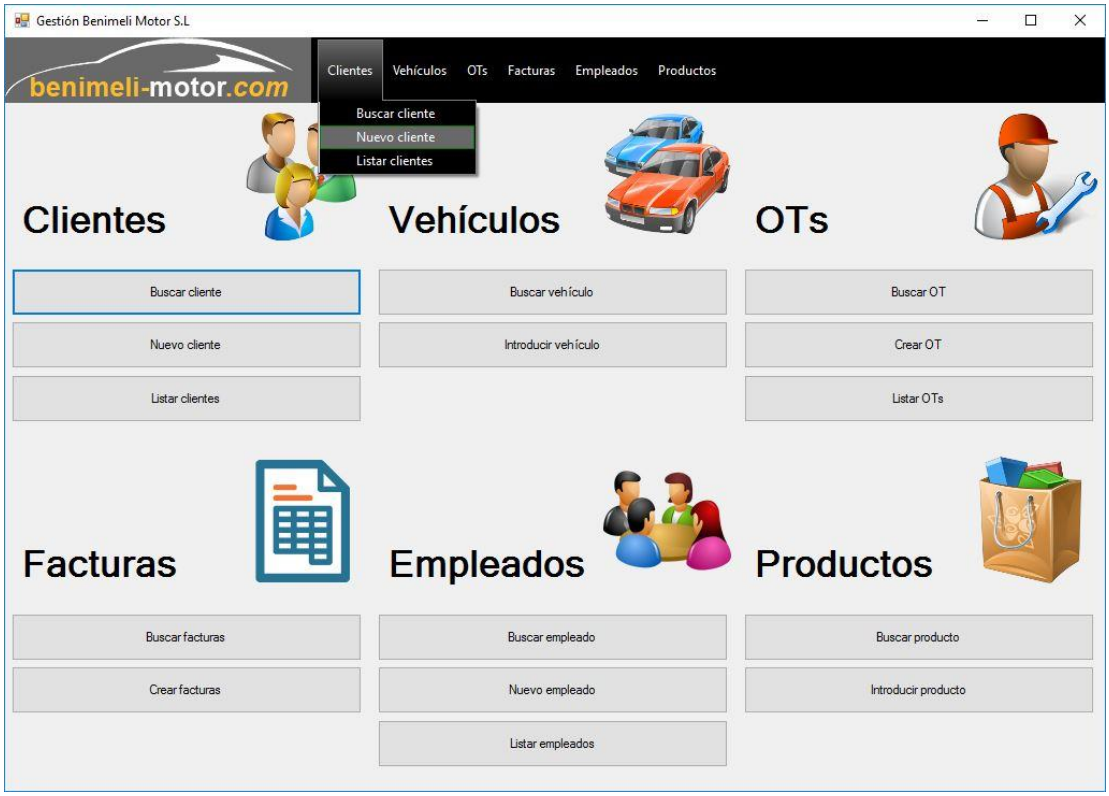


Ilustración 14 - Pantalla principal

En caso de que el usuario tenga nivel de técnico, no tendrá acceso a muchas de las pantallas, en las que solo se puede acceder siendo administrador. La siguiente imagen muestra la pantalla principal vista por un técnico.

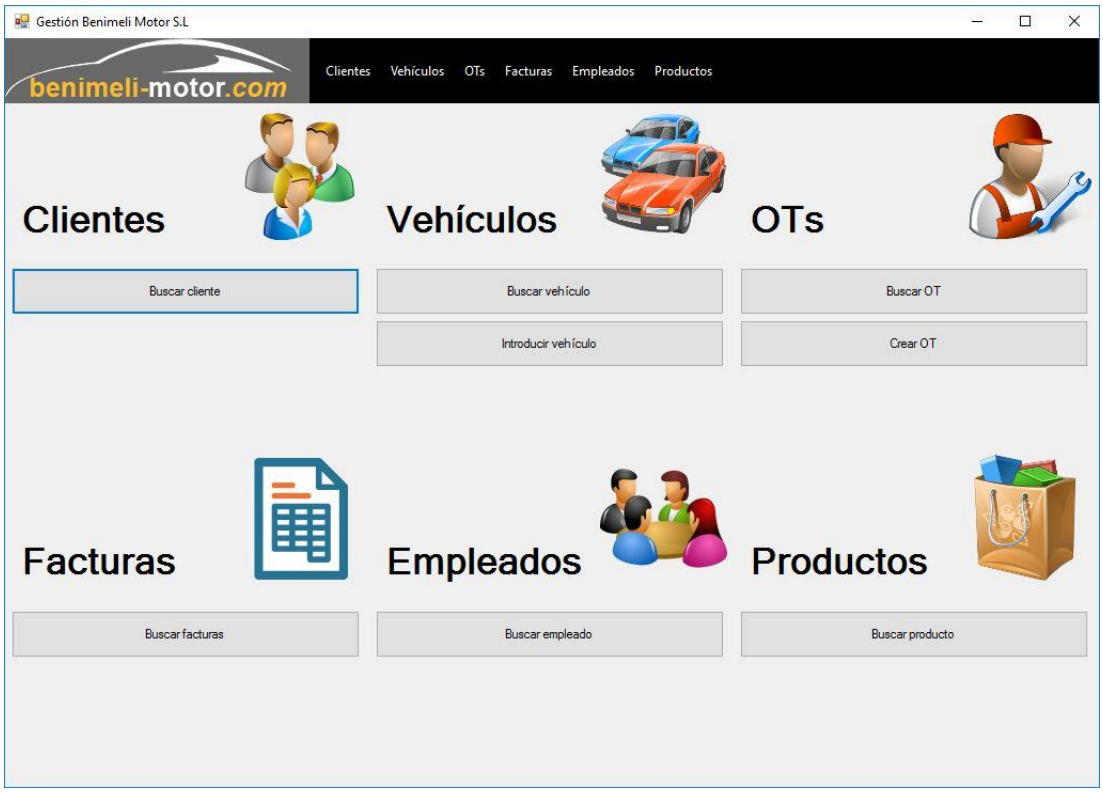


Ilustración 15 - Pantalla principal vista por un técnico

Si accedemos a la pantalla de buscar cliente, podemos buscar mediante id, nombre, apellidos o teléfono. No es necesario escribir el campo entero para buscar, pero es posible que aparezcan varios resultados si no se pone exacto. En esta pantalla (y en todas las que son similares) se ha añadido el botón de visualizar cliente, que no estaba en el *mockup* inicial. Este será el único botón visible para un empleado con el nivel de técnico.

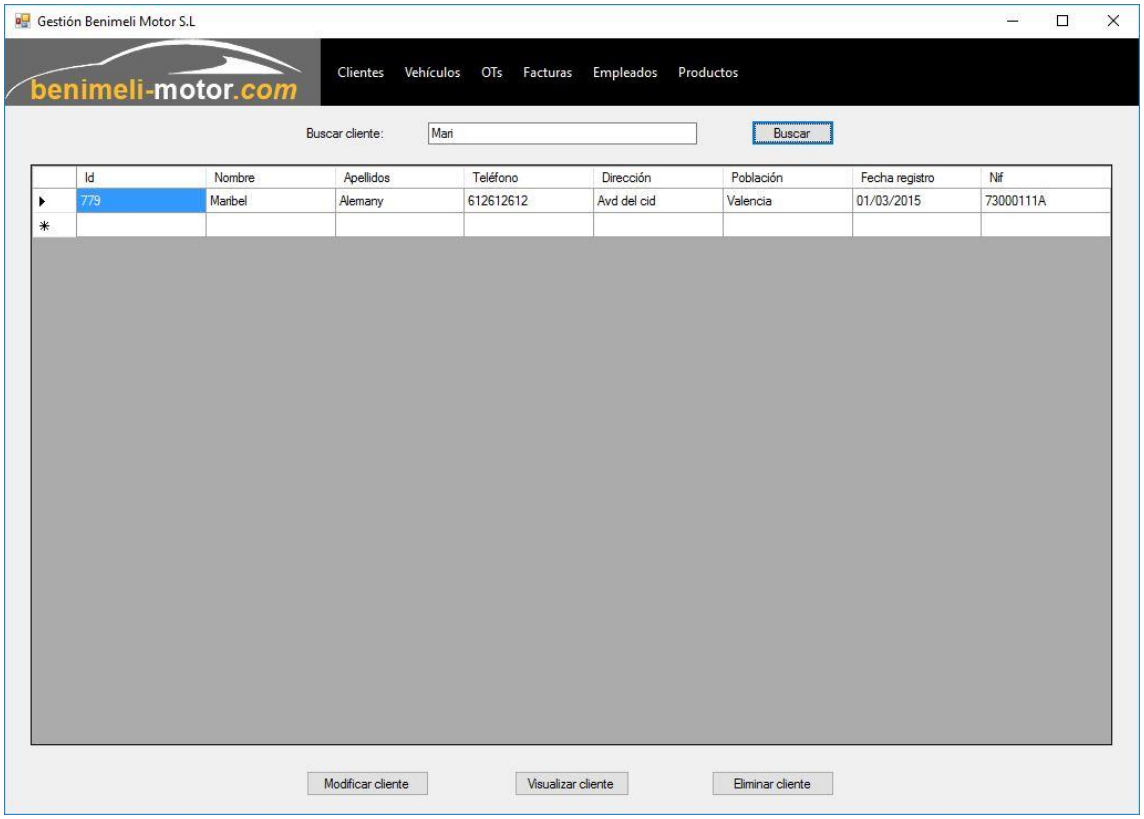


Ilustración 16 - Buscar cliente

Aquí la pantalla de buscar cliente vista por un técnico.

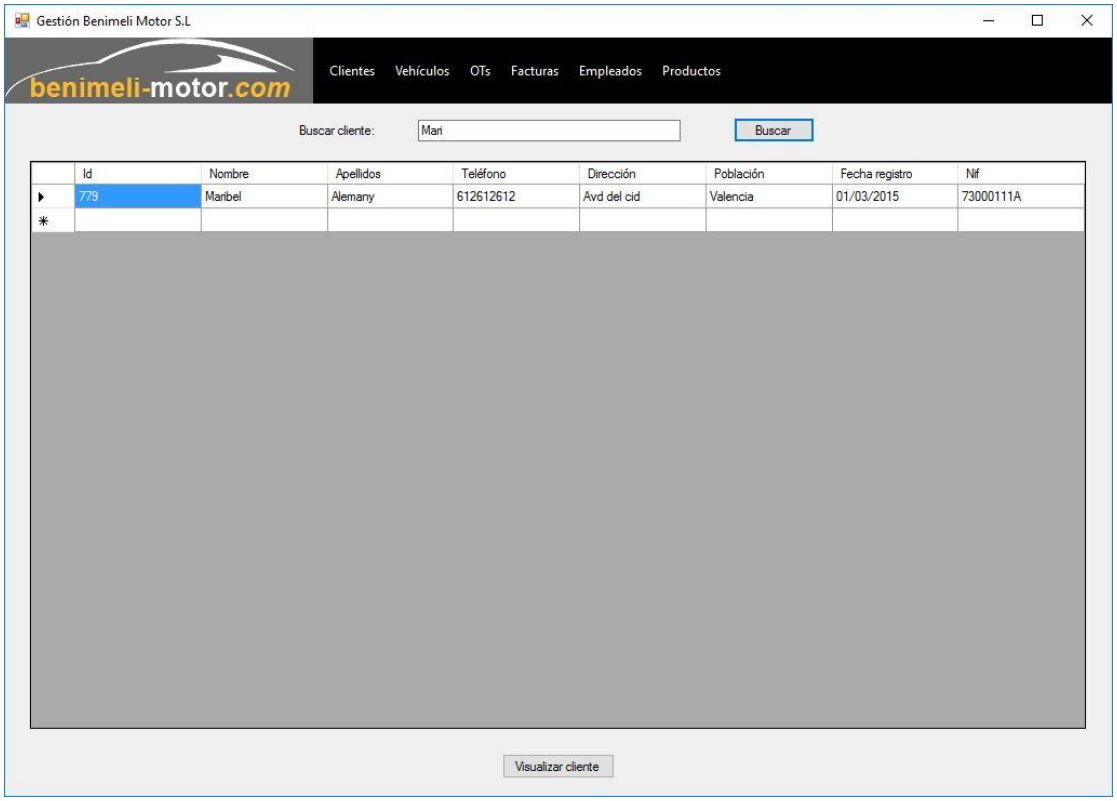


Ilustración 17 - Buscar cliente visto por un técnico

Si introducimos un nuevo cliente, tenemos que rellenar todos los campos. En el caso de la id, pulsando el botón generar se asignará un id automático en relación a los existentes.

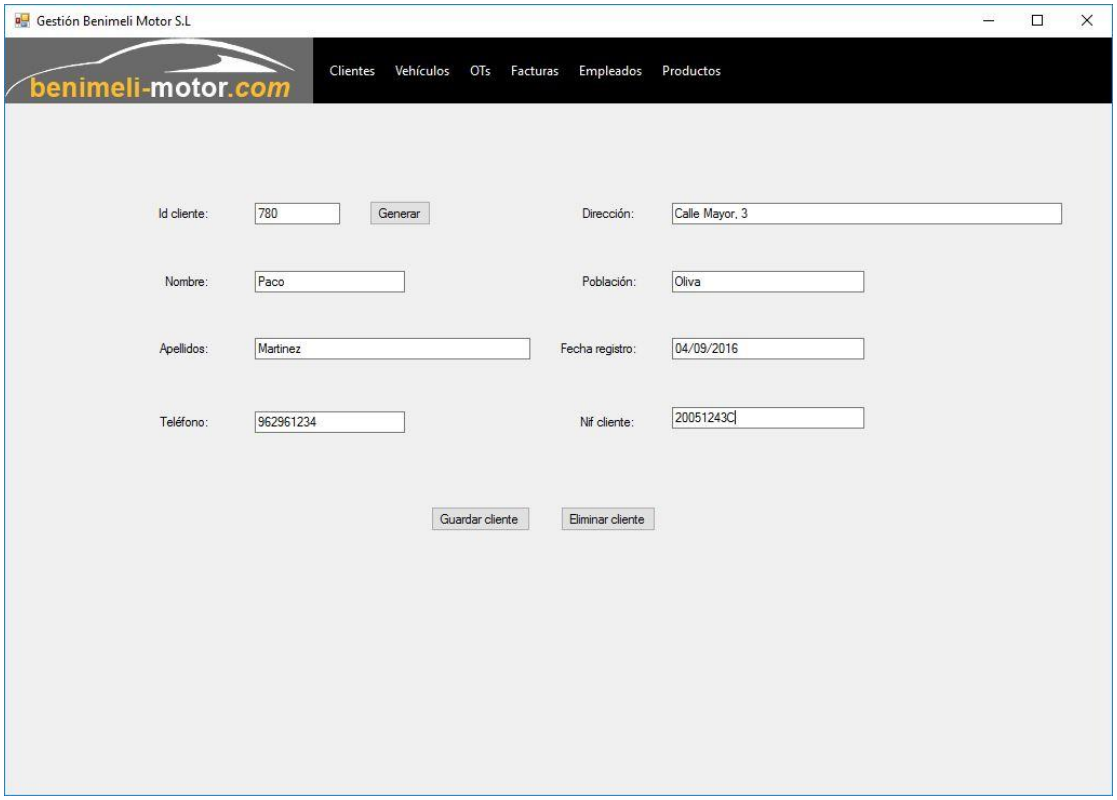


Ilustración 18 - Nuevo cliente

Cuando un cliente se introduce correctamente, se deshabilitan todos los campos y aparecen un campo de confirmación en la parte inferior.

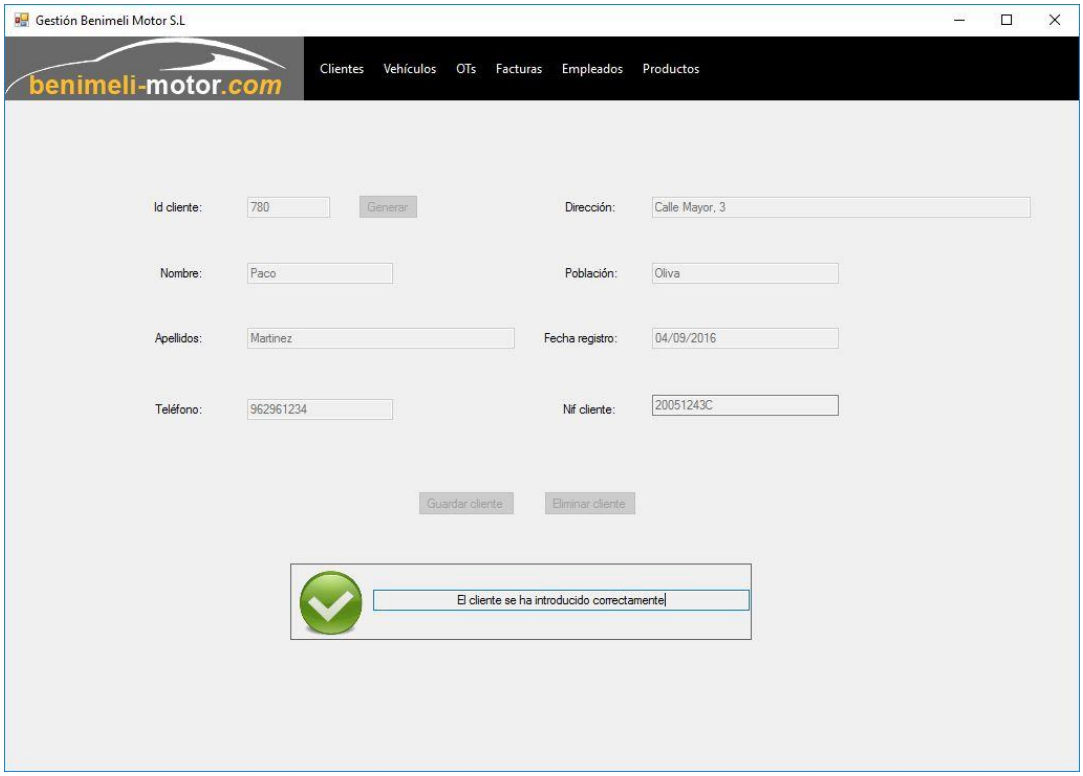


Ilustración 19 - Cliente introducido correctamente

Si accedemos a listar clientes, aparece un listado con todos los clientes registrados.

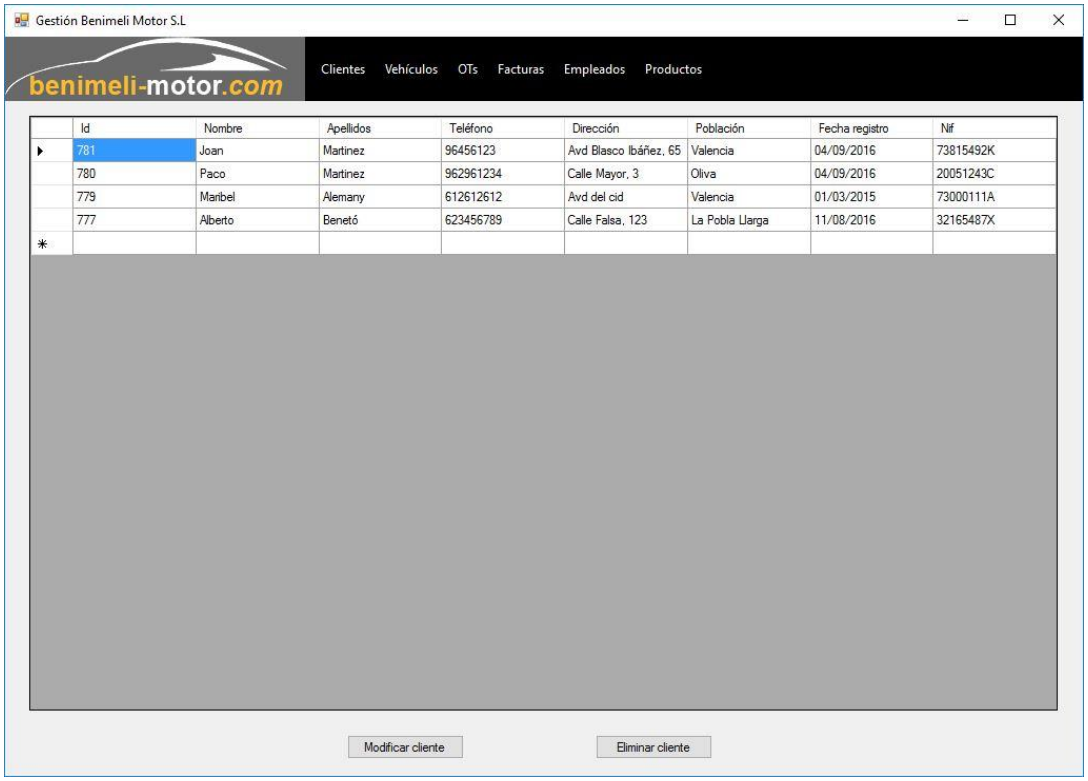


Ilustración 20 - Listar clientes

Si seleccionamos un cliente y pulsamos en modificar cliente (ya sea desde la pantalla de buscar cliente o desde listar clientes) accederemos al formulario de modificación. Tiene los mismos campos que de cliente nuevo con la excepción de que el campo id cliente no se puede modificar.

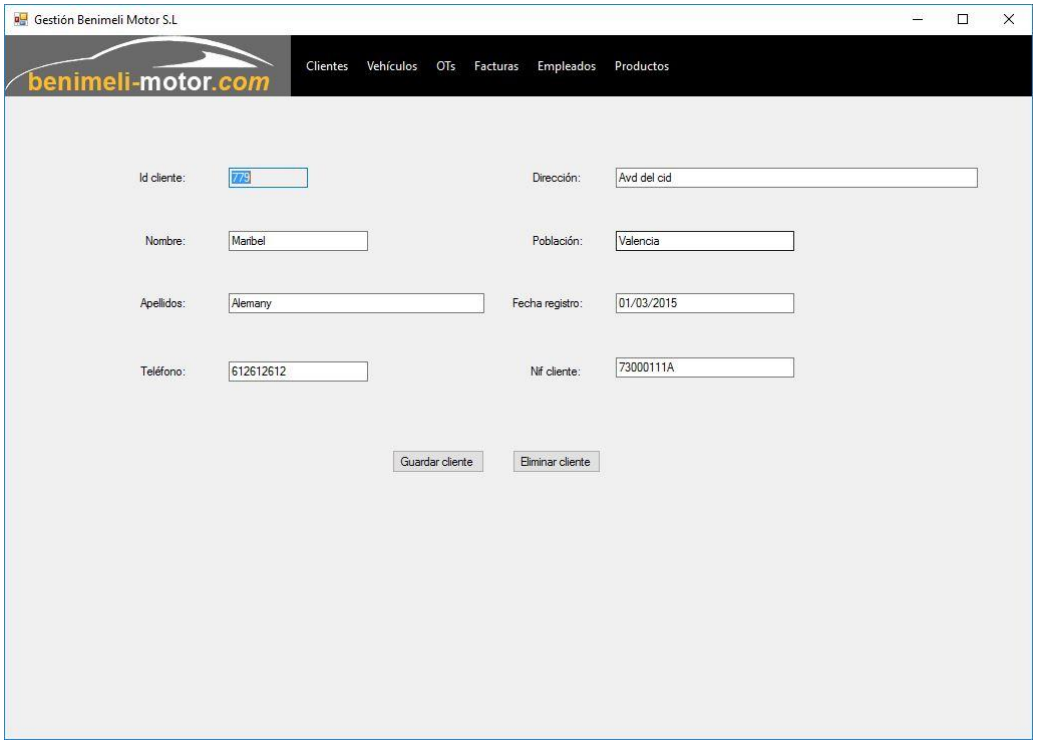


Ilustración 21 - Modificar cliente

En el caso de seleccionar un cliente y pulsar en visualizar cliente, accederemos al modo visualización, donde únicamente podemos visualizar los datos del cliente.

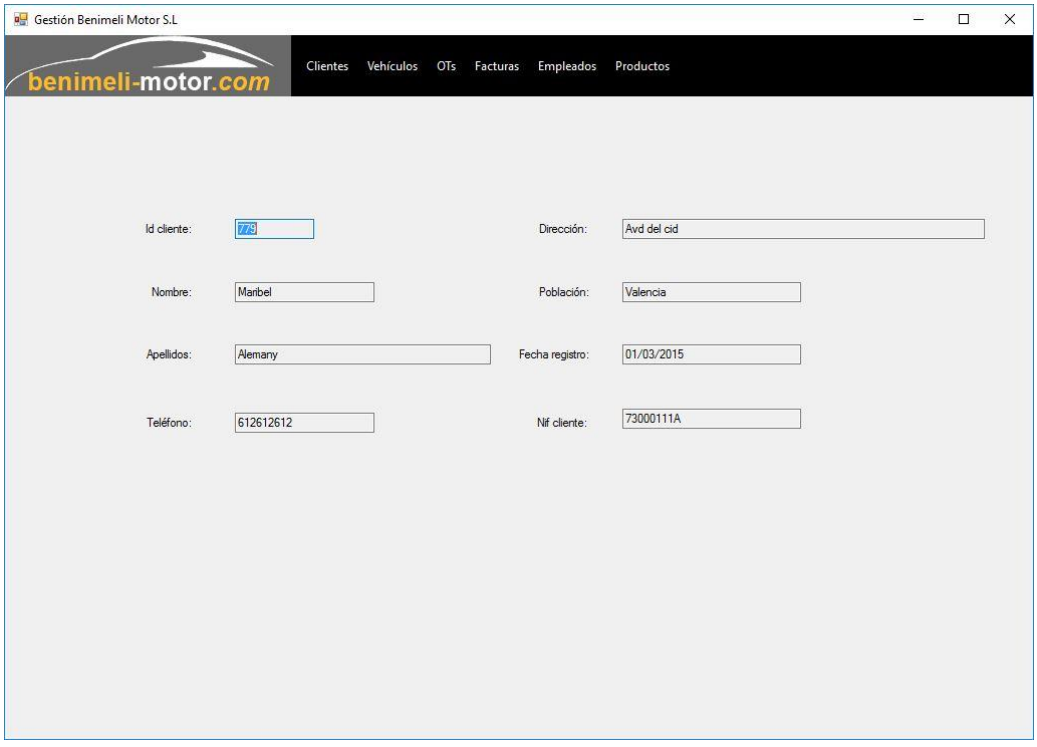


Ilustración 22 - Visualizar cliente

En la pantalla de buscar vehículo, podemos buscar un vehículo mediante matrícula, número de bastidor o id de cliente.

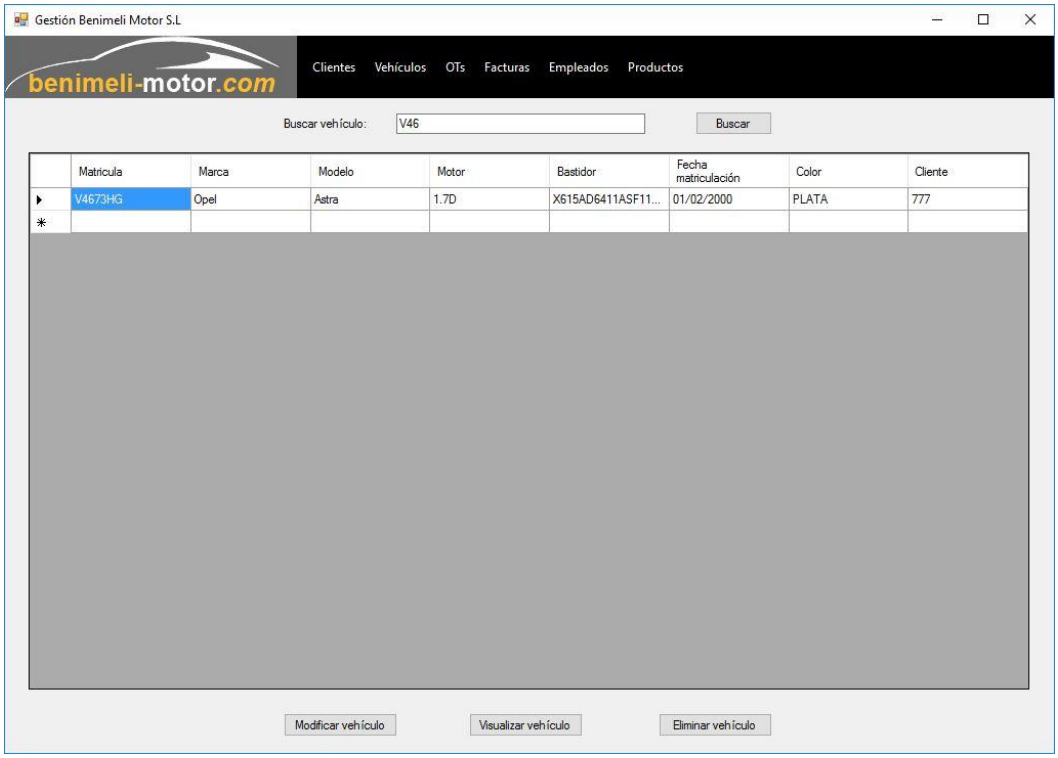


Ilustración 23 - Buscar vehículo

Igual que al modificar un cliente, si accedemos a modificar un vehículo, no se podrá modificar la matrícula de este.

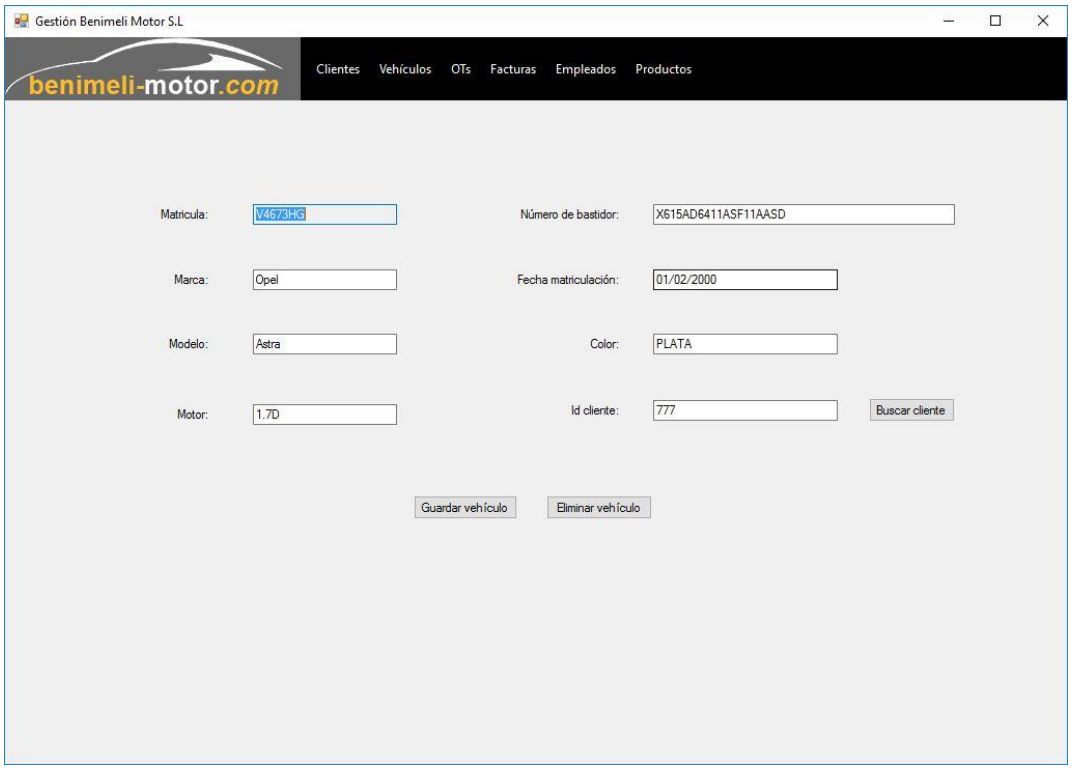


Ilustración 24 - Modificar vehículo

En la pantalla de introducir un nuevo vehículo tendremos que rellenar todos los campos.

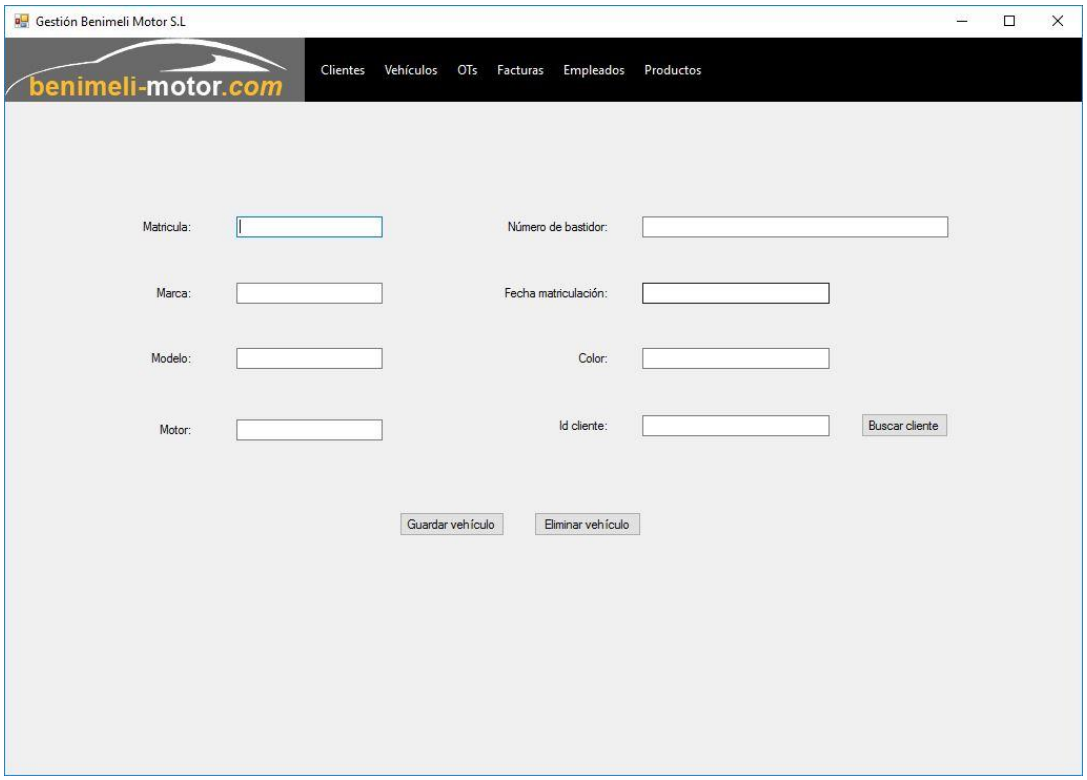


Ilustración 25 - Nuevo vehículo

En la pantalla anterior, si pulsamos el botón de buscar cliente, se abrirá una ventana auxiliar para buscar el cliente deseado. Luego pulsando en insertar cliente, se añadirá al formulario.

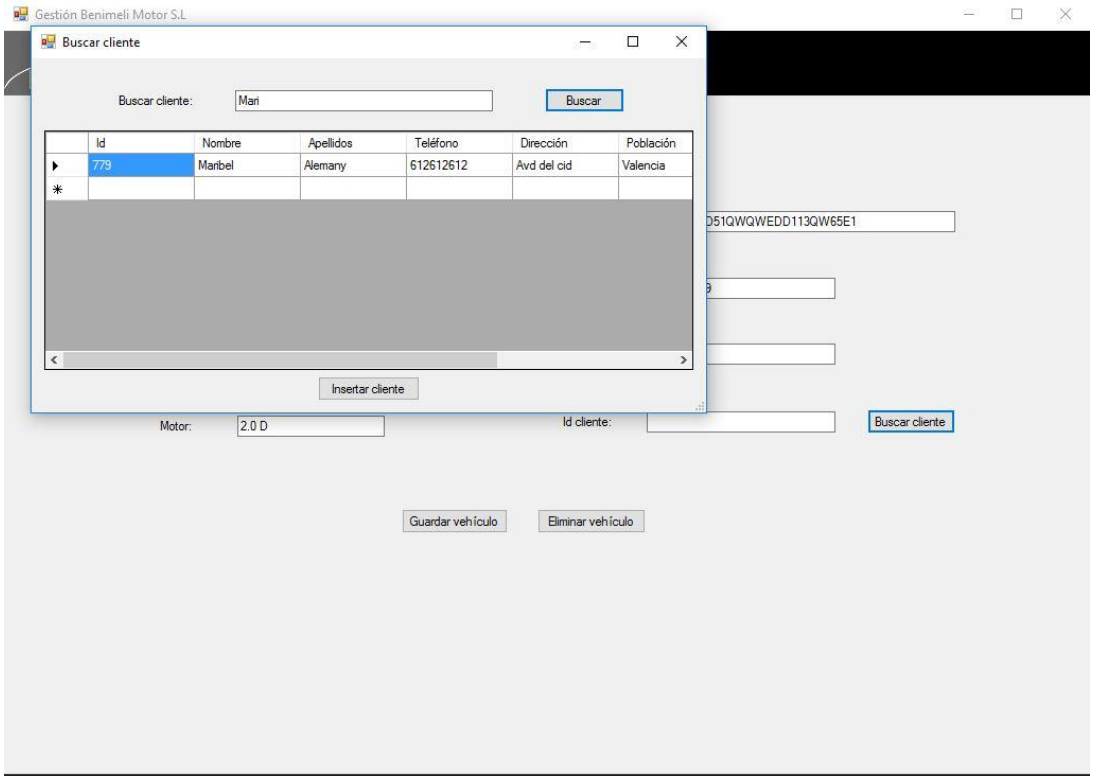


Ilustración 26 - Ventana de buscar cliente

Una vez introducido el vehículo, se deshabilitan los campos y aparece un mensaje de confirmación.

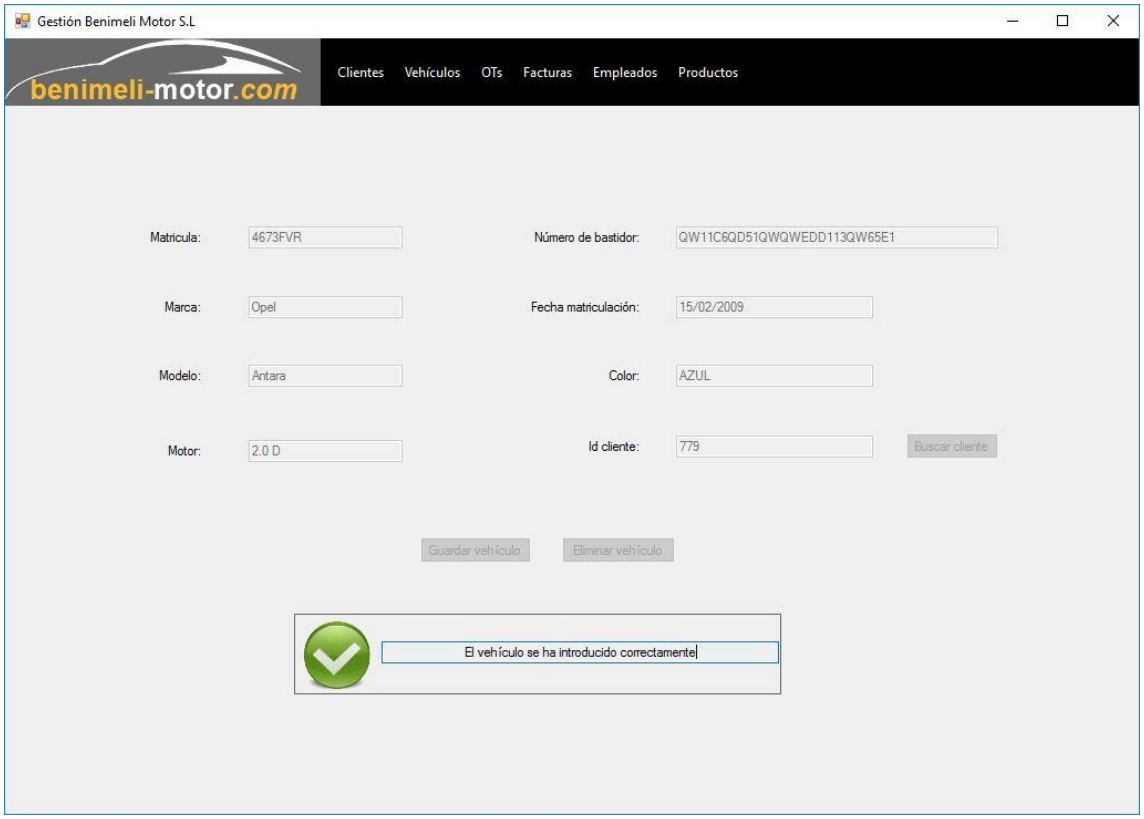


Ilustración 27 - Vehículo introducido correctamente

En el formulario de crear OT tendremos que rellenar todos los campos. El número de OT se generará automáticamente pulsando el botón y la matrícula del vehículo se inserta buscando el vehículo en la ventana auxiliar. Cuando se inserte la matrícula del vehículo, se insertará automáticamente su id de cliente. En el grid de trabajos tendremos que detallar lo que el empleado debe realizar al coche en cuestión.

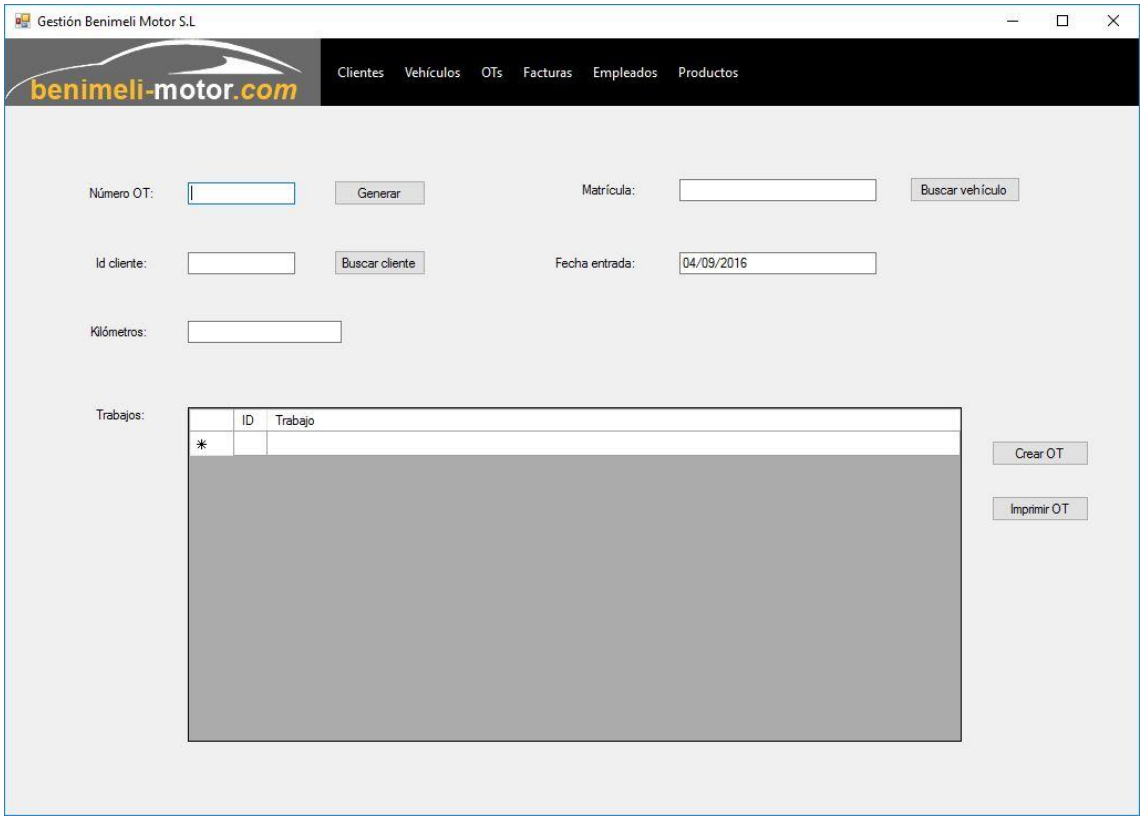


Ilustración 28 - Crear OT

Si la OT se crea correctamente, aparecerá un mensaje de confirmación.

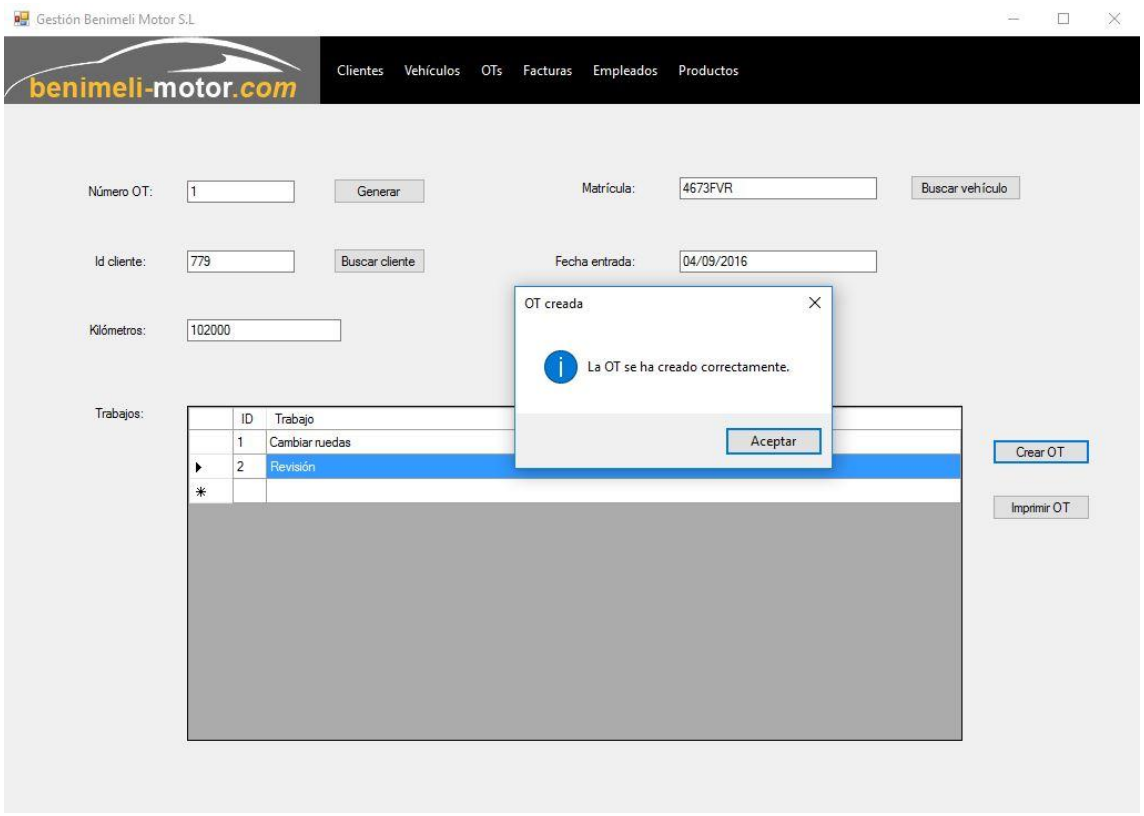


Ilustración 29 - OT creada correctamente

Después de crear una OT se abrirá el formulario de visualización de la OT, desde la que podemos crear un informe pulsando en imprimir OT.

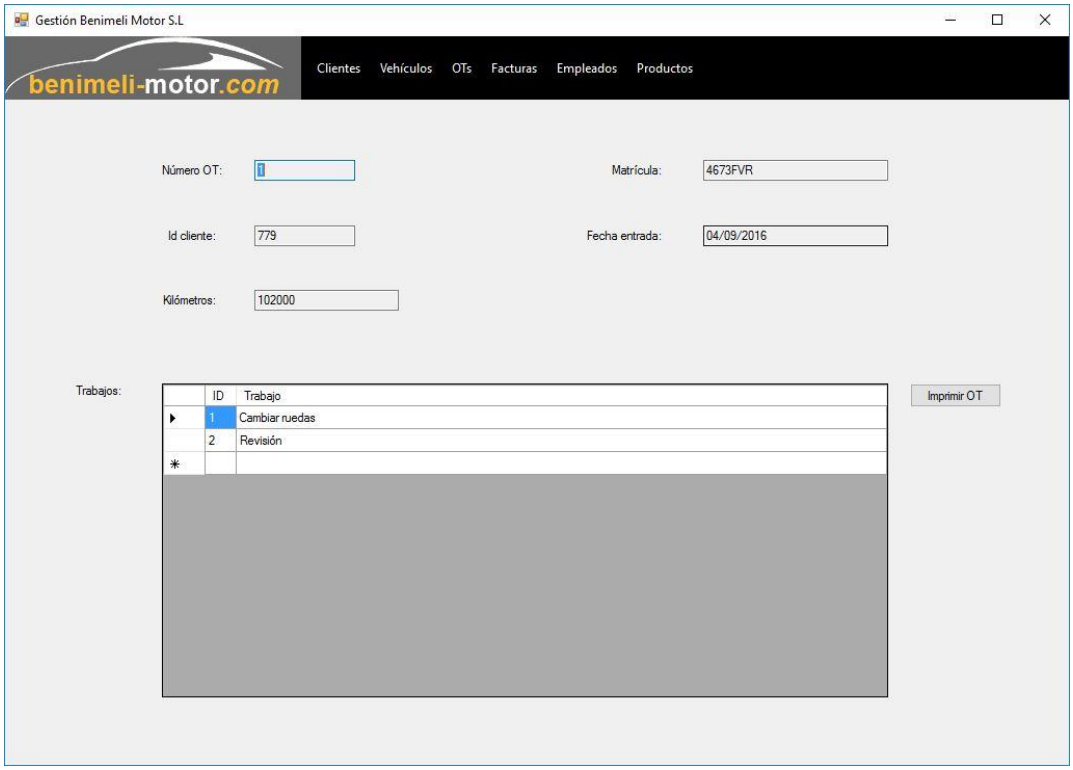


Ilustración 30 - Visualizar OT

Cuando se crea el informe, se abre en el visor de informes desde el que se puede imprimir y guardar como PDF.

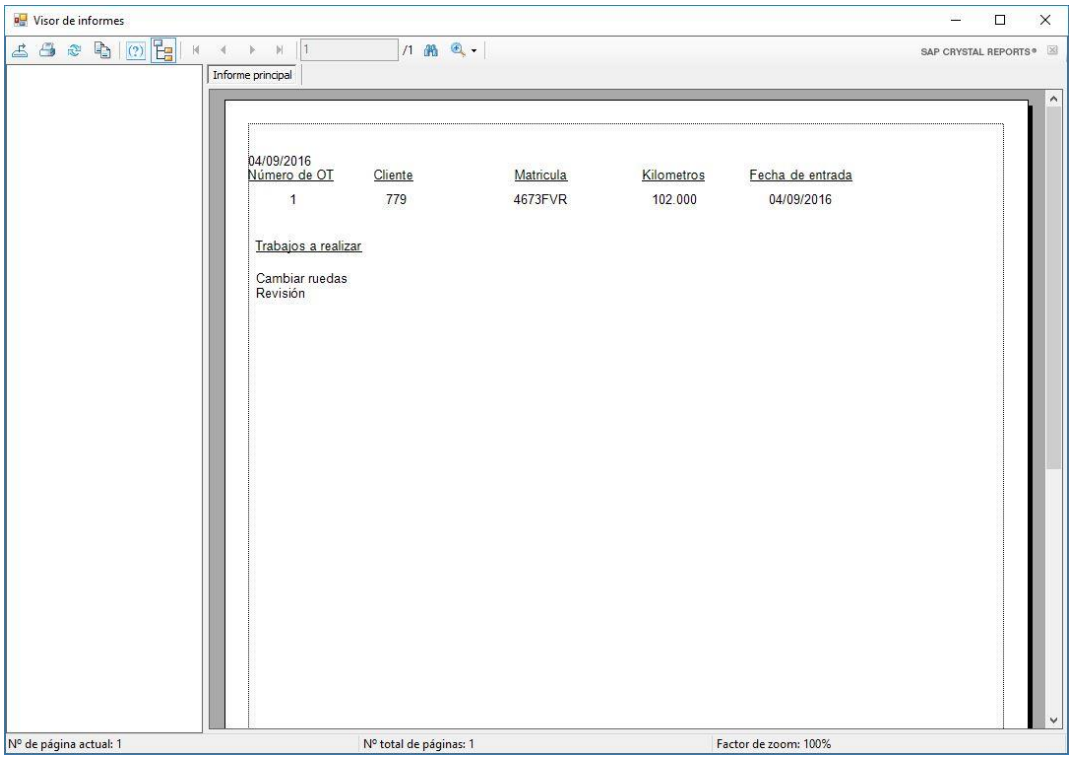


Ilustración 31 - Informe de OT

Si accedemos a crear una factura, tendremos que generar un número de factura y buscar la OT de la que se desee crear una factura. Cuando se inserte la OT, se actualizarán automáticamente el id cliente y la matrícula. En la fecha de factura se insertará el día actual automáticamente. En el grid de productos, insertaremos una referencia y se rellenarán automáticamente el nombre de producto y su precio. Si cambiamos la cantidad de producto, se actualizará su precio.

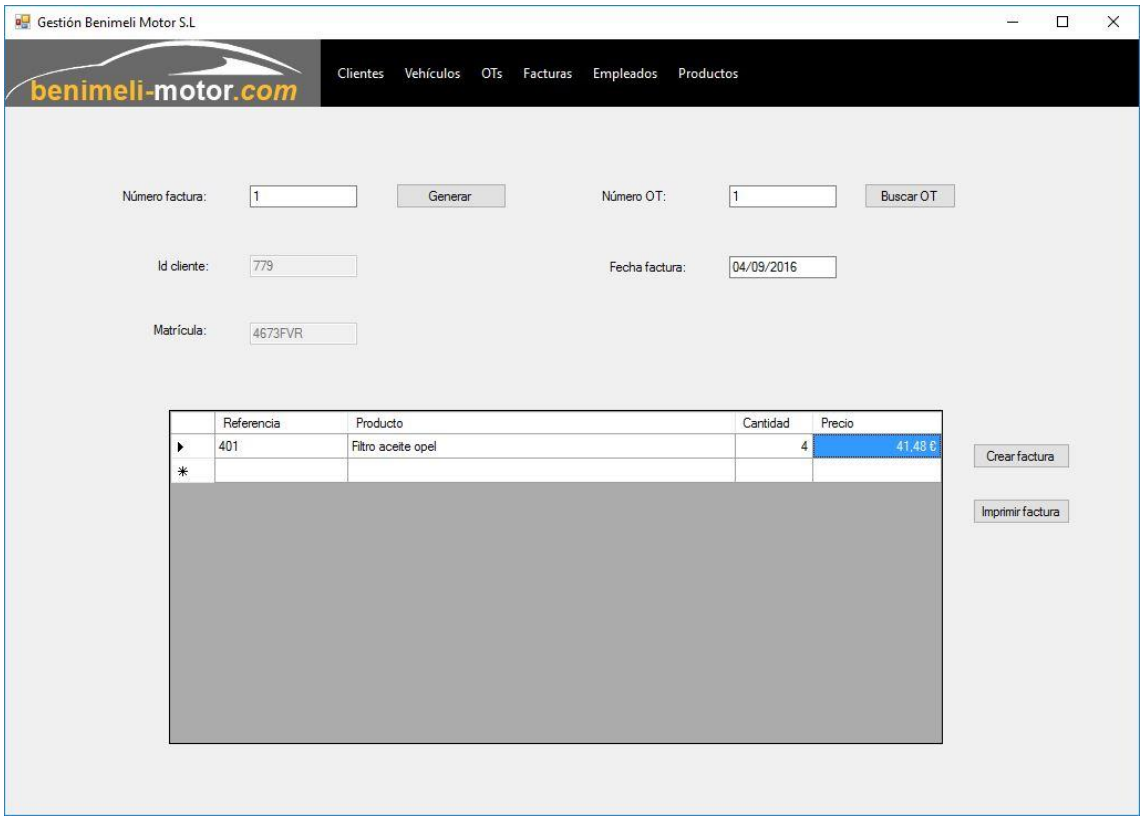


Ilustración 32 - Crear factura

Se puede buscar un empleado mediante su id, nombre, apellidos, teléfono o email. Aunque rara vez se usará esta pantalla, sabiendo la cantidad de empleados de la empresa, será mucho más eficiente abrir directamente el listado de todos los empleados.

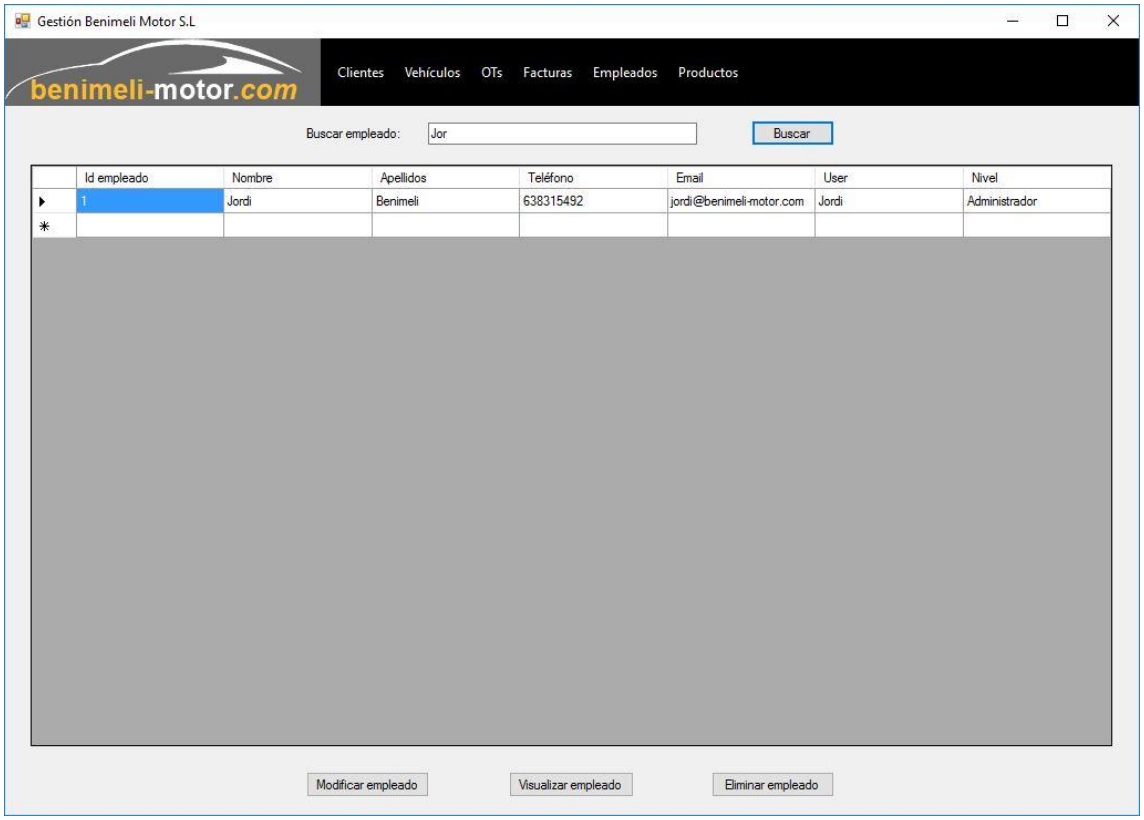


Ilustración 33 - Buscar empleado

Si deseamos modificar un empleado, podremos modificar todos los campos excepto su id. En el caso de la contraseña de usuario, se deberá dejar en blanco si no se desea modificar.

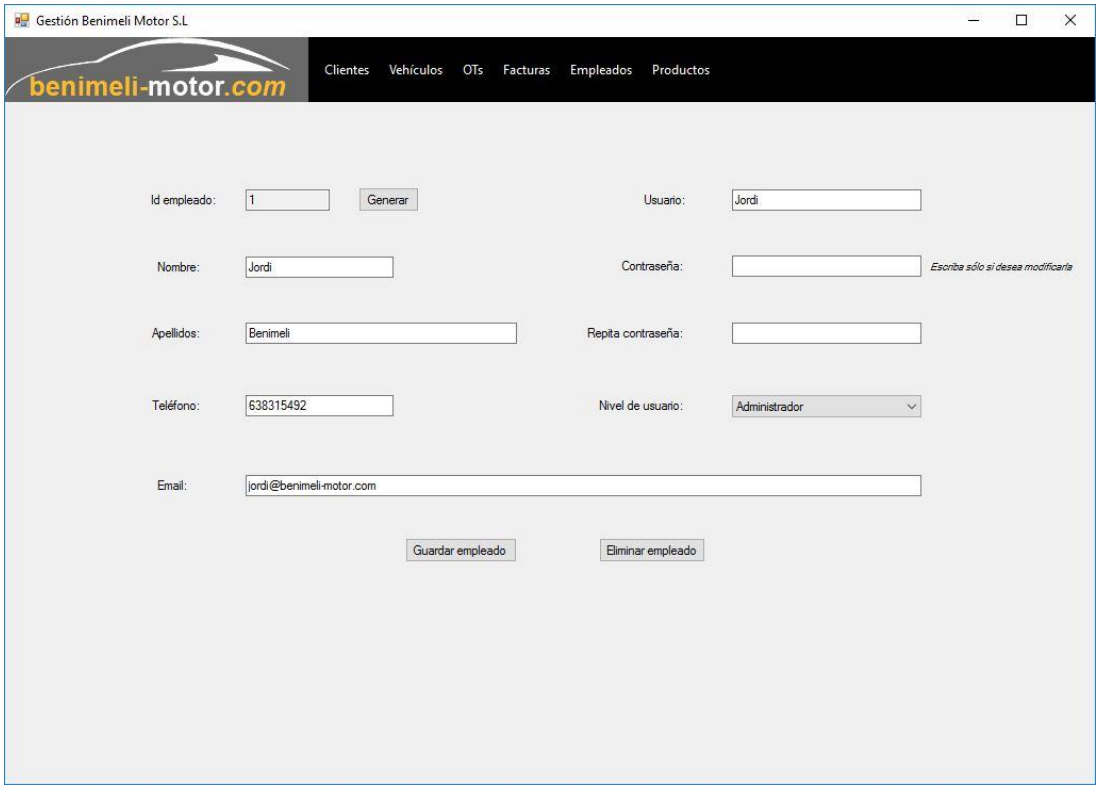


Ilustración 34 - Modificar empleado

En el formulario de nuevo empleado (al igual que en modificar empleado), si las contraseñas introducidas no coinciden aparecerá una cruz roja al lado del campo repetir contraseña. Si se intenta guardar el empleado mientras las contraseñas no coinciden, saltará un mensaje de error.

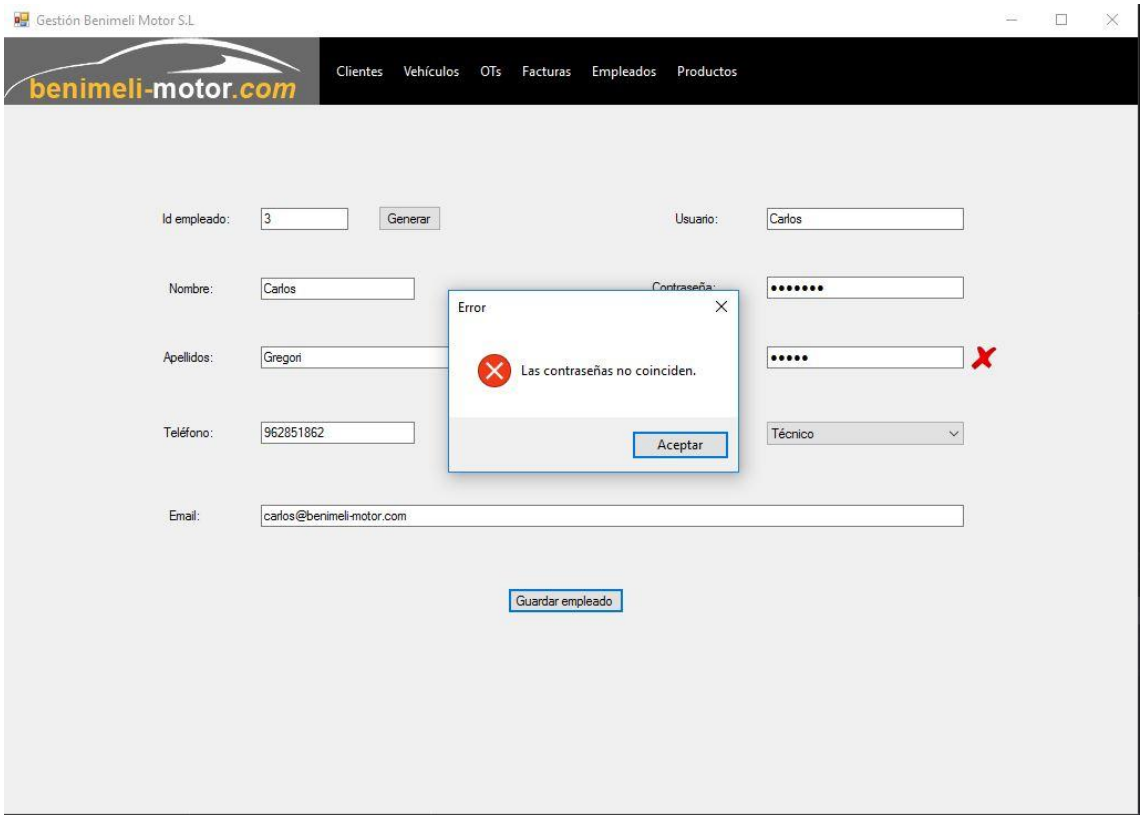


Ilustración 35 - Error al insertar empleado

Si las contraseñas coinciden, aparecerá un *tick* verde al lado del campo de repetir contraseña. Si el empleado se introduce correctamente, aparecerá un mensaje de confirmación.

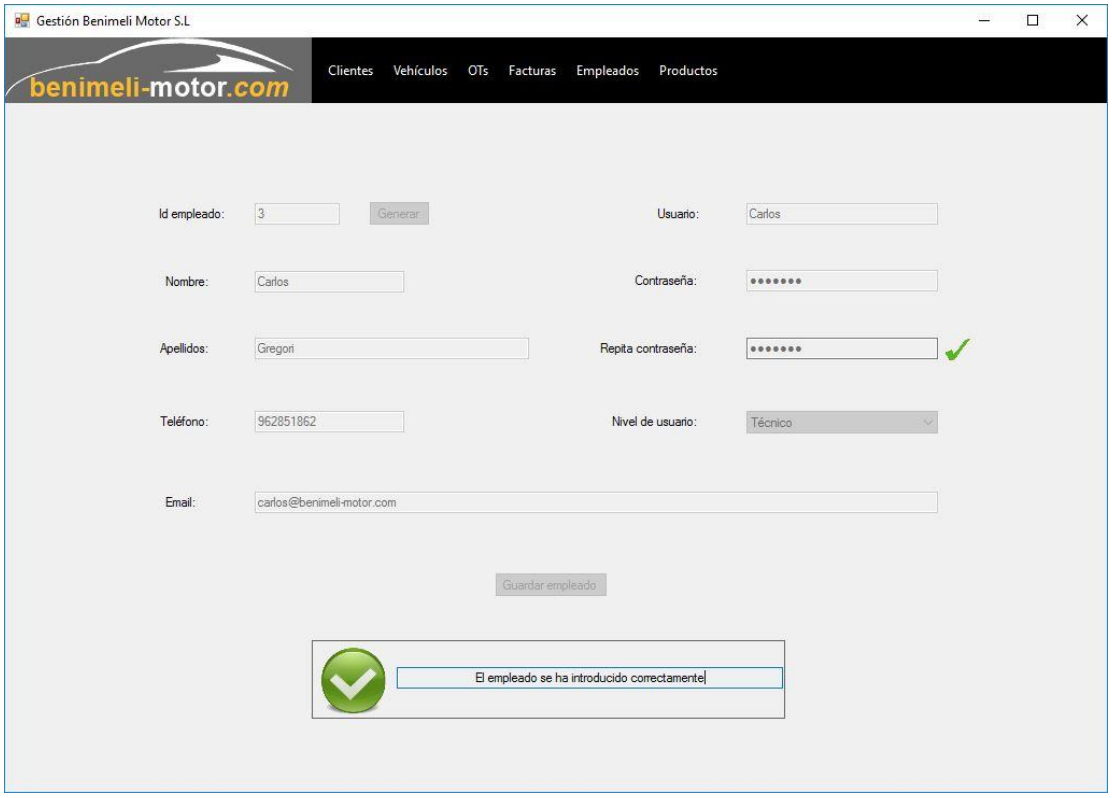


Ilustración 36 - Empleado introducido correctamente

Si se procede a eliminar un registro en la aplicación (sea un vehículo, un cliente, un empleado o un producto), aparecerá una ventana para confirmar la operación.

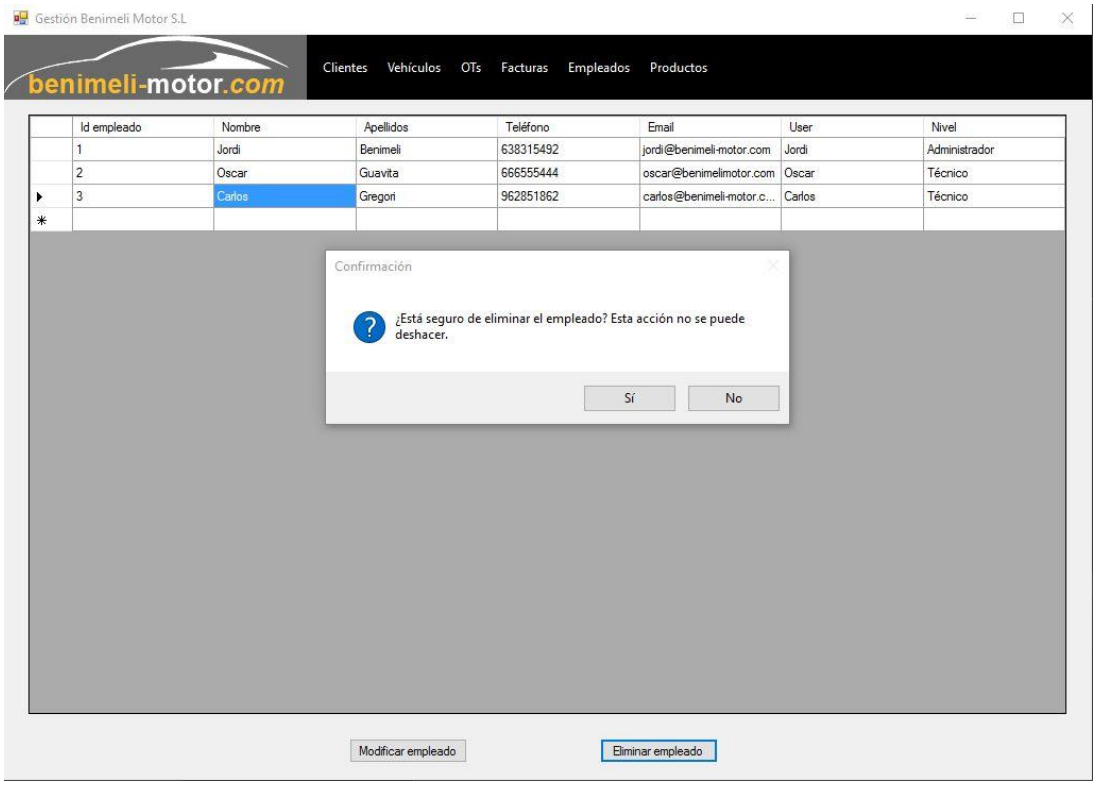


Ilustración 37 - Confirmación para eliminar empleado

En el formulario de introducir producto, tendremos que rellenar todos los campos. Al introducir la referencia deseada se deberá pulsar el botón de comprobar para que el programa verifique que la referencia no existe en la base de datos. En el *comboBox* familia se escogerá el tipo de recambio que se va a introducir.

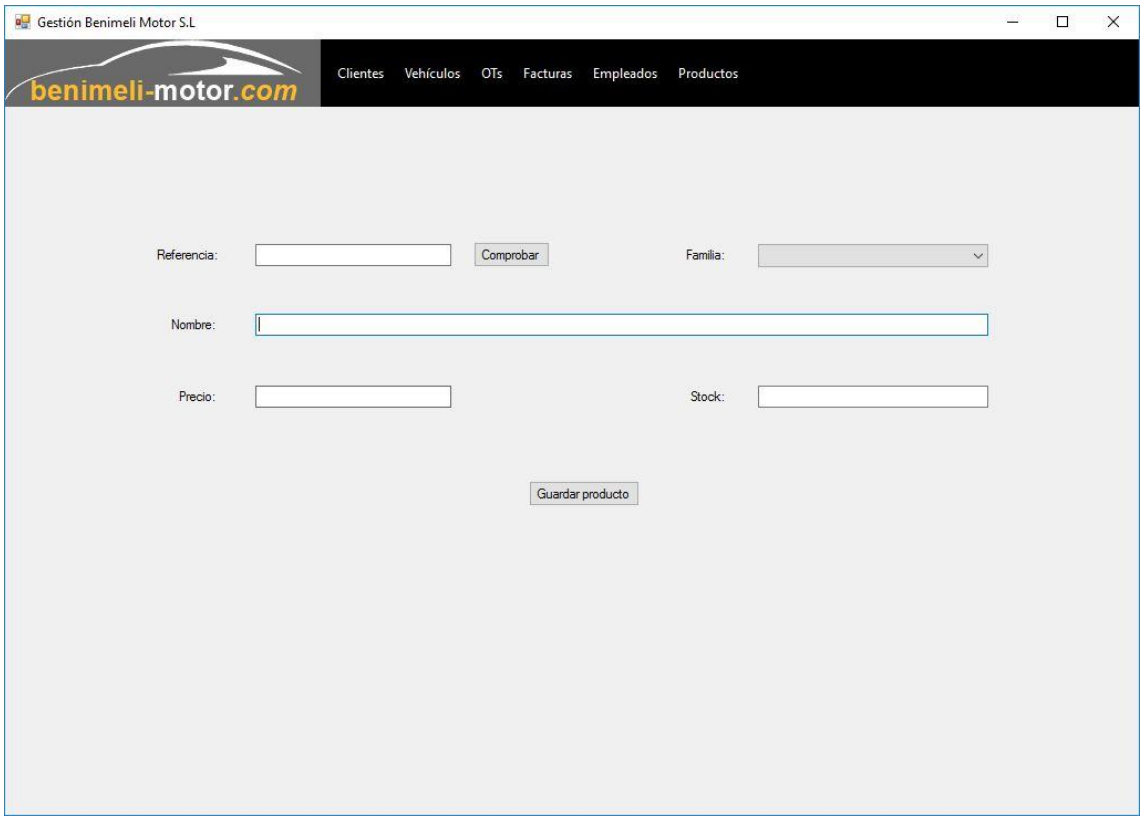


Ilustración 38 - Nuevo producto

6. Ampliaciones futuras

Durante el desarrollo del presente software, se han implementado los procesos incluidos en el software al que va a sustituir. A medida que se iba implementando, aparecían nuevas funcionalidades o procesos que podrían ser de gran utilidad en un futuro. Algunas de las funcionalidades son ideas propias y otras son peticiones expresas de la empresa.

Las funcionalidades previstas de incluir a corto plazo son:

- Modificar los informes de OT y facturas, los actuales se han desarrollado únicamente para probar la funcionalidad. Se debe unificar con el modelo que se usa actualmente en la empresa.
- Incluir un *comboBox* de los empleados en el formulario para crear una orden de trabajo. De esta forma, se tiene constancia de a qué trabajador se le ha asignado la OT.
- Una vez implementada la funcionalidad anterior, se creará un listado personal de OTs diarias en función al usuario que esté logueado en la aplicación. Así todos los trabajadores tienen acceso a un rápido resumen de su jornada laboral.
- Se incluirá también la posibilidad de poner una imagen al crear un producto. Esto es muy útil para comparar la pieza afectada con el recambio necesario.
- Se añadirá a los productos un campo de stock mínimo. De esta forma, también podremos generar un listado de productos por debajo de su stock mínimo facilitando los pedidos de material.

A medio plazo se incluirán las funcionalidades descritas a continuación:

- En el apartado de facturas, se creará un nuevo tipo de factura de proveedores. En él se registrarán todos los pedidos de material realizados por el taller para posteriormente poder extraer facturaciones trimestrales o anuales según proveedor.
- Al entrar en un empleado, se añadirá un campo de rendimiento. Este campo relacionará las horas de trabajo facturadas con las horas de trabajo reales de cada empleado.

Por último, una vez se hayan implementado todas las mejoras descritas, se procederá a sincronizar esta aplicación con la aplicación de control de trabajo que utiliza la empresa. Este sistema, utiliza un lector de código de barras con las siguientes funcionalidades:

- Al inicio y final de la jornada laboral, cada empleado debe pasar su código para que quede registrada su entrada y salida.
- Cuando se inicia y finaliza el trabajo detallado en una OT, el empleado debe pasar el código de inicio de trabajo y el de fin de trabajo para que quede registrado exactamente la mano de obra que se ha necesitado.

Sincronizando ambos sistemas, al crear una nueva factura, dispondremos de la mano de obra que se ha utilizado en cada trabajo sin necesidad de tener que introducir a mano mirando la otra aplicación. Además, al estar registrado las horas de trabajo de cada empleado, el campo de rendimiento de cada uno será más preciso y fiable.

Esta última mejora será sin duda la más difícil de realizar, razón por la que será la última en implementarse. No obstante, he podido contactar con la empresa desarrolladora de este sistema y me han ofrecido toda la ayuda y documentación que necesite de su sistema para llevar a cabo mi objetivo.

7.Conclusiones

Una vez finalizada la implementación inicial, se ha conseguido crear un software que cumple con los requisitos establecidos en la fase de análisis. Por lo cual, podemos afirmar que se ha cumplido el objetivo.

Se ha intentado automatizar, en la medida de lo posible, la introducción de datos en los formularios. Además, al estar dirigido a una empresa concreta, se ha adaptado la forma de trabajar del sistema a su actual forma de trabajar. Por tanto, la curva de aprendizaje del nuevo sistema será baja.

Como ya he comentado en el capítulo 1.2, tengo este proyecto en mente desde que empecé el Grado en Ingeniería Informática. Por tanto, he cogido este TFG con muchas ganas, porque el desarrollo de la aplicación no va acabar cuando esté entregado, sino que voy a continuar desarrollando todo lo que la empresa de mi familia necesite implementar.

Esta ha sido mi primera aplicación desarrollada desde 0 en C# y estoy orgulloso de haberla podido llevar a cabo. Tiene mucho margen de mejora y esto seguro que cuando implemente todo lo descrito saldrán nuevas funcionalidades para incluir.

En el apartado técnico, he aprendido mucho usando Visual Studio y he mejorado mi destreza y rapidez desarrollando a medida que iba aprendiendo nuevos métodos, atajos de teclado y formas de programar que incluye este gran IDE. Además, he aprendido la importancia de usar un sistema de control de versiones. Porque aparte de estar perfectamente integrado en Visual Studio, me ha permitido revertir muchas veces diversos cambios en las clases. También me salvó cuando en una ocasión tuve que recuperar todo el proyecto al corromperlo moviéndolo a otra ubicación.

8.Bibliografía

https://es.wikipedia.org/wiki/Transact-SQL

http://www.foro.lospillaos.es/conectarse-a-base-de-datos-desde-otro-ordenador-vt7620.html

https://es.wikipedia.org/wiki/Requisito_funcional

http://www.ecured.cu/Requisitos_no_funcionales

Ingenieria.de.Software.-.Ian.Sommerville.7ma.Edicion

http://www.humbertocervantes.net/homepage/itzamna/DOCUMENTACION/Doc3.html

http://users.dcc.uchile.cl/~psalinas/uml/casosuso.html

https://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_software

http://es.slideshare.net/AntonioMoreno22/presentacin-uml

http://ltuttini.blogspot.com.es/2011/07/archivos-de-configuracion-una.html

http://joseluisgarciab.blogspot.com.es/2014/09/programacion-en-3-capas.html

http://ltuttini.blogspot.com.es/2010/05/login-usando-password-con-hash.html

https://es.wikipedia.org/wiki/Programaci%C3%B3n_por_capas

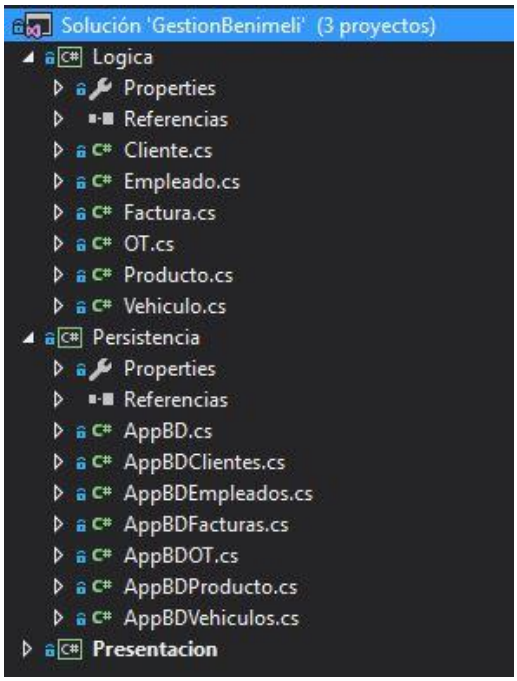
https://www.codejobs.biz/es/blog/2014/01/28/la-programacion-por-capas

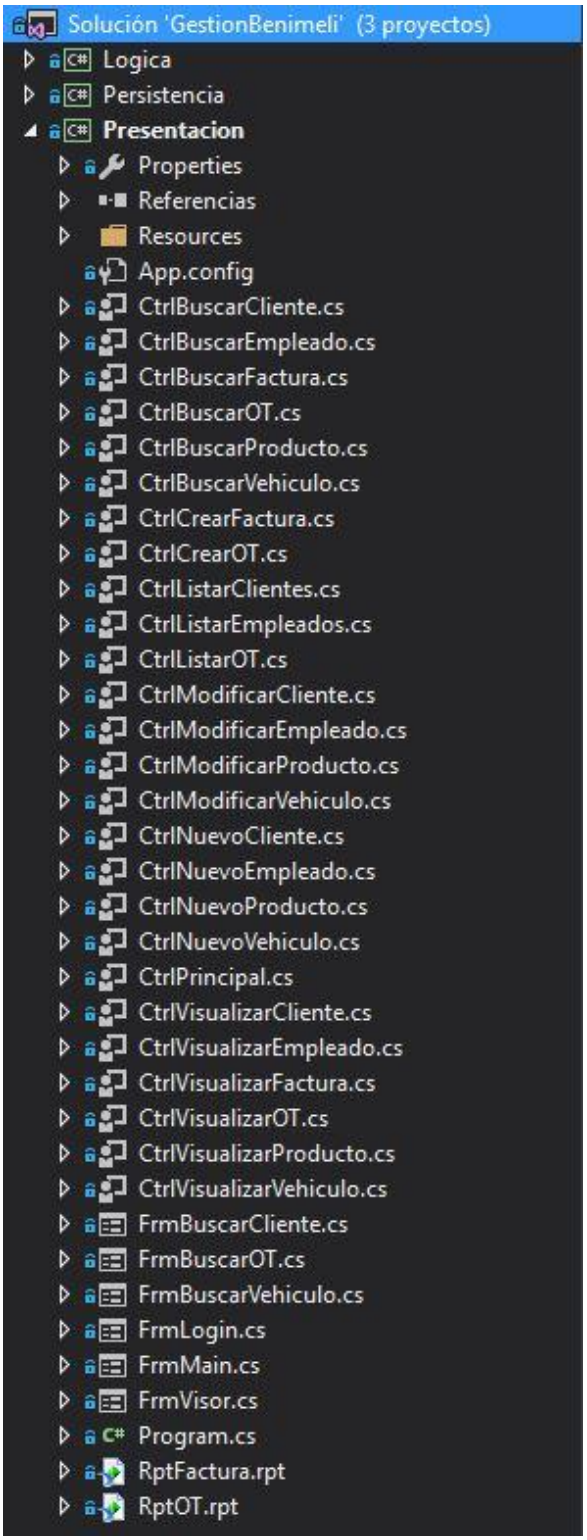
https://en.wikipedia.org/wiki/.NET_Framework

9.Anexos

9.1 Anexo 1 – Clases del proyecto

En el presente proyecto se han utilizado las siguientes clases repartidas en tres capas:





Para descargar el proyecto y revisar sus clases se puede acceder desde el siguiente enlace:

https://github.com/Benimeli/GestionBenimeli

9.2 Anexo 2 – Base de datos

En el siguiente script se puede observar las distintas tablas de la base de datos y sus relaciones. Así como los dos tipos de tablas que se tuvieron que definir.

```
USE [gestionBenimeli]
GO

/****** Object:  Table [dbo].[CLIENTES]    ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[CLIENTES](
    [idCliente] [int] NOT NULL,
    [nombre] [nvarchar](50) NOT NULL,
    [apellidos] [nvarchar](50) NOT NULL,
    [telefono] [nvarchar](50) NOT NULL,
    [direccion] [nvarchar](50) NOT NULL,
    [poblacion] [nvarchar](50) NOT NULL,
    [fechaRegistro] [nvarchar](50) NOT NULL,
    [nifCliente] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_CLIENTES] PRIMARY KEY CLUSTERED
(
    [idCliente] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
/****** Object:  Table [dbo].[DETALLE_FACTURA]    ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[DETALLE_FACTURA](
    [numFactura] [int] NOT NULL,
    [referencia] [nvarchar](50) NOT NULL,
    [cantidad] [decimal](18, 2) NOT NULL
) ON [PRIMARY]

GO
/****** Object:  Table [dbo].[DETALLE_TRABAJO_OT]    ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[DETALLE_TRABAJO_OT](
    [numOT] [int] NOT NULL,
    [trabajo] [nvarchar](max) NOT NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]

GO


/****** Object:  Table [dbo].[EMPLEADOS]    ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[EMPLEADOS](
    [idEmpleado] [int] NOT NULL,
    [nombre] [nvarchar](50) NOT NULL,
    [apellidos] [nvarchar](max) NOT NULL,
    [telefono] [nvarchar](50) NOT NULL,
    [email] [nvarchar](50) NULL,
    [user] [nvarchar](50) NOT NULL,
    [password] [nvarchar](50) NOT NULL,
    [nivel] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_EMPLEADOS] PRIMARY KEY CLUSTERED
(
    [idEmpleado] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]

GO
/****** Object:  Table [dbo].[FACTURAS]    ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[FACTURAS](
    [numFactura] [int] NOT NULL,
    [idCliente] [int] NOT NULL,
    [matricula] [nvarchar](50) NOT NULL,
    [numOT] [int] NOT NULL,
    [fechaFactura] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_FACTURAS] PRIMARY KEY CLUSTERED
(
    [numFactura] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

```
/****** Object:  Table [dbo].[OT]    ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[OT](
    [numOT] [int] NOT NULL,
    [idCliente] [int] NOT NULL,
    [matricula] [nvarchar](50) NOT NULL,
    [kilometros] [int] NOT NULL,
    [fechaEntrada] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_OT] PRIMARY KEY CLUSTERED
(
    [numOT] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
/****** Object:  Table [dbo].[PRODUCTOS]    ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[PRODUCTOS](
    [referencia] [nvarchar](50) NOT NULL,
    [nombre] [nvarchar](50) NOT NULL,
    [familia] [nvarchar](50) NOT NULL,
    [precio] [decimal](18, 2) NOT NULL,
    [stock] [int] NOT NULL,
 CONSTRAINT [PK_PRODUCTOS] PRIMARY KEY CLUSTERED
(
    [referencia] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]


/****** Object:  Table [dbo].[VEHICULOS]    ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[VEHICULOS](
    [matricula] [nvarchar](50) NOT NULL,
    [marca] [nvarchar](50) NOT NULL,
    [modelo] [nvarchar](50) NOT NULL,
    [motor] [nvarchar](50) NOT NULL,
    [numBastidor] [nvarchar](50) NOT NULL,
    [fechaMatriculacion] [nvarchar](50) NOT NULL,
    [color] [nvarchar](50) NOT NULL,
    [idCliente] [int] NOT NULL,
 CONSTRAINT [PK_Vehiculos] PRIMARY KEY CLUSTERED
(
    [matricula] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
ALTER TABLE [dbo].[DETALLE_TRABAJO_OT]  WITH CHECK ADD  CONSTRAINT [FK_DETALLE_TRABAJOS_OT_OT] FOREIGN KEY([numOT])
REFERENCES [dbo].[OT] ([numOT])
GO
ALTER TABLE [dbo].[DETALLE_TRABAJO_OT] CHECK CONSTRAINT [FK_DETALLE_TRABAJOS_OT_OT]
GO
ALTER TABLE [dbo].[FACTURAS]  WITH CHECK ADD  CONSTRAINT [FK_FACTURAS_CLIENTES] FOREIGN KEY([idCliente])
REFERENCES [dbo].[CLIENTES] ([idCliente])
GO
ALTER TABLE [dbo].[FACTURAS] CHECK CONSTRAINT [FK_FACTURAS_CLIENTES]
GO
ALTER TABLE [dbo].[FACTURAS]  WITH CHECK ADD  CONSTRAINT [FK_FACTURAS_OT] FOREIGN KEY([numOT])
REFERENCES [dbo].[OT] ([numOT])
GO
ALTER TABLE [dbo].[FACTURAS] CHECK CONSTRAINT [FK_FACTURAS_OT]
GO
ALTER TABLE [dbo].[FACTURAS]  WITH CHECK ADD  CONSTRAINT [FK_FACTURAS_VEHICULOS] FOREIGN KEY([matricula])
REFERENCES [dbo].[VEHICULOS] ([matricula])
GO
ALTER TABLE [dbo].[FACTURAS] CHECK CONSTRAINT [FK_FACTURAS_VEHICULOS]
GO
ALTER TABLE [dbo].[OT]  WITH CHECK ADD  CONSTRAINT [FK_OT_CLIENTES] FOREIGN KEY([idCliente])
REFERENCES [dbo].[CLIENTES] ([idCliente])
GO
ALTER TABLE [dbo].[OT] CHECK CONSTRAINT [FK_OT_CLIENTES]
GO
ALTER TABLE [dbo].[OT]  WITH CHECK ADD  CONSTRAINT [FK_OT_VEHICULOS] FOREIGN KEY([matricula])
REFERENCES [dbo].[VEHICULOS] ([matricula])
GO
ALTER TABLE [dbo].[OT] CHECK CONSTRAINT [FK_OT_VEHICULOS]
GO
ALTER TABLE [dbo].[VEHICULOS]  WITH CHECK ADD  CONSTRAINT [FK_VEHICULOS_CLIENTES] FOREIGN KEY([idCliente])
REFERENCES [dbo].[CLIENTES] ([idCliente])
GO
ALTER TABLE [dbo].[VEHICULOS] CHECK CONSTRAINT [FK_VEHICULOS_CLIENTES]
GO
```

```
/****** Object:  UserDefinedTableType [dbo].[DetalleFactura]    ******/
CREATE TYPE [dbo].[DetalleFactura] AS TABLE(
    [id] [int] NOT NULL,
    [referencia] [varchar](50) NOT NULL,
    [cantidad] [numeric](9, 2) NOT NULL,
    PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH (IGNORE_DUP_KEY = OFF)
)
GO
/****** Object:  UserDefinedTableType [dbo].[DetalleTrabajo]    ******/
CREATE TYPE [dbo].[DetalleTrabajo] AS TABLE(
    [id] [int] NOT NULL,
    [trabajo] [varchar](50) NOT NULL,
    PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH (IGNORE_DUP_KEY = OFF)
)
GO
```

La siguiente imagen muestra los diferentes procedimientos almacenadas que se han utilizado para comunicar la capa de datos (o persistencia) con la base de datos.