UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Diseño e implementación de un sistema para gestionar el alquiler de vehículos

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Maksym Chmutov Derevianko

Tutor: José Ángel Carsí Cubel

2018 - 2019

Resumen

Este trabajo tiene como objetivo el diseño e implementación de un sistema software que permita realizar la gestión de vehículos por parte de los usuarios finales desde cualquiera de sus oficinas de alquiler de vehículos. Para ello, se ha partido de la necesidad existente por parte de cada uno de los usuarios finales que van a interactuar con el sistema, de manera que las necesidades de cada usuario vienen expresadas en un diagrama de casos de uso.

Desde el punto de vista del diseño, se ha optado por la realización de un software en el que se separan claramente las distintas componentes que lo componen. Por dicha razón, se ha optado por la creación de una solución con una arquitectura multicapa de tres niveles formada por las capas de presentación, lógica de negocio y persistencia separando así las componentes que forman el sistema. Adicionalmente, tras considerar las necesidades definidas en el diagrama de casos de uso se ha diseñado el diagrama de clases que muestra las entidades que pueden existir dentro de la aplicación y la relación entre estas. Esto ha permitido separar la aplicación en módulos, creando así una mejor organización, facilitando su implementación y garantizando un mejor mantenimiento de la aplicación en el futuro. En lo referente a las tecnologías utilizadas, para crear dicha solución se han utilizado el controlador de versiones Team Services, el entorno de desarrollo integrado (IDE) Visual Studio junto al lenguaje de programación C#, las tecnologías Windows Presentation Foundation (WPF) y Entity Framework.

Como resultado, tras las fases de análisis, diseño, implementación y pruebas se ha obtenido una solución que cumple con los requisitos establecidos por las necesidades de los usuarios finales.

Palabras clave: diseño, implementación, sistema, gestión, vehículos, Microsoft, WPF, C#, EF.

Abstract

This work aims to design and implement a software system that allows the management of vehicles by end users from any of the car rental branch offices. For this, the needs of each of the end users that will interact with the system have been considered, so that all those needs are expressed in the use case diagram.

From the point of view of design, the software has been created with the aim of having all the different components that compose it clearly separated. For this reason, the solution has been created with an architecture of three layers which are the presentation, the business logic and the persistence layers which separate those components. Additionally, after considering the needs defined in the use case diagram, the class diagram has been designed and it shows the entities that may exist within the application and the relationship between them. This has allowed the application to be separated into modules, thus creating a better organization, facilitating its implementation and guaranteeing a better maintenance of the application in the future. The technologies that have been used for creating this solution are Team Services version control, the Visual Studio integrated development environment (IDE) together with the C # programming language, Windows Presentation Foundation (WPF) and Entity Framework.

As a result, after the phases of analysis, design, implementation and testing a solution that meets the requirements established by the needs of the end users has been created.

Keywords: design, implementation, management, system, vehicles, Microsoft, WPF, C#, EF.

Tabla de contenidos

Tabla de contenido

1. Introducción

Visual Studio es actualmente uno de los entornos de desarrollo integrado más conocidos del mundo. Este entorno ofrece la posibilidad de utilizar y combinar múltiples herramientas para el desarrollo de software a medida.

En este documento se expondrá como se ha llevado a cabo la realización del proyecto "Vehicle Rental System", un software de gestión de vehículos que podría ser utilizado por una empresa de alquiler de vehículos que tuviese múltiples oficinas por todo el mundo. Para ello, se ha hecho bastante énfasis en las buenas prácticas durante todas las fases del proceso de creación del software y en aspectos como la seguridad o la modularidad que ha permitido dividir la aplicación en partes más pequeñas también llamadas módulos lo que ofrece múltiples ventajas que se expondrán más adelante.

Por último, detallar también que la idea de "Vehicle Rental System" no viene siendo la de lanzar un nuevo sistema de gestión de vehículos al mercado, sino que esta estará orientada al ámbito educacional siendo un ejemplo de solución real para los alumnos que cursen la asignatura de "Ingeniería del Software".

1.1 Motivación

Las principales motivaciones para elegir este contenido y orientación del trabajo realizado son la posibilidad de crear un sistema en base a unos requisitos previamente proporcionados y con las tecnologías actualmente utilizadas en el sector tecnológico.

Adicionalmente, surge la oportunidad de poner en práctica los conocimientos adquiridos en la rama de ingeniería del software y de potenciar las capacidades de diseño, implementación y pruebas de sistemas, considerando además la ventaja que ello pueda ofrecer para el futuro laboral.

1.2 Objetivos

La creación de un sistema de gestión de vehículos utilizando las buenas prácticas de la ingeniería del software. De esta forma, el objeto de este también será el de servir como una solución real de referencia para los estudiantes.

Para lograr ambos objetivos, hay que tener en cuenta el sector o ámbito al que va dirigido dicho sistema para así proporcionar un software de calidad.

El trabajo está dividido en dos grandes objetivos que se dividen en subobjetivos:

- **Diseñar** el software en base a los requisitos proporcionados y teniendo en cuenta el ámbito al que va destinado.
    - Diseñar el diagrama de clases.
- **Implementar** todos los módulos en los que se subdivide y forman la aplicación final.
    - Generar e implementar todas las clases.
    - Subdividir la aplicación en tres capas.
    - Implementar la capa de persistencia de datos y testear.

- Implementar la capa de la lógica de negocio y testear.
- Implementar la capa de la interfaz gráfica y testear.
- Documentar toda la aplicación

1.3 Estructura

La memoria está estructurada en diferentes apartados, estando formada principalmente por 8 capítulos que permiten profundizar en múltiples aspectos dentro de los procesos seguidos durante el desarrollo de la aplicación.

En el capítulo 2 se realiza un estudio de mercado en el que se muestran dos productos distintos de los ofrecidos en dicho mercado y se especifica la funcionalidad que ofrecen. Por último, se indican las mejoras que implementa el software desarrollado en este proyecto con respecto a estos dos.

En el capítulo 3 se comentan los requisitos de la aplicación por medio de una descripción del producto. Adicionalmente, se considera el entorno al que va dirigido la aplicación, los usuarios finales y las limitaciones para su creación.

El capítulo 4 se centra en la fase de análisis. En esta, se realizará un diagrama de casos de uso en el que se detallan las posibles interacciones de los actores con el sistema y también se mostrará el diagrama de clases realizado en dicha fase.

En el capítulo 5 se habla sobre el estilo arquitectónico que implementa la aplicación, de la razón de haber elegido este y se proporcionará un diagrama de clases más detallado que en la fase anterior. Adicionalmente, se muestra la transición entre las distintas ventanas de la aplicación.

El capítulo 6 se basa en los detalles de la implementación del programa. Entre estos se incluyen las tecnologías utilizadas, la estructura de la solución y detalles sobre el código implementado en las distintas capas.

En el capítulo 7 se muestran las distintas técnicas abordadas para comprobar el correcto funcionamiento de cada capa que forma la aplicación. También se hablará la oportunidad que brinda Visual Studio para la realización de pruebas unitarias y sobre las pruebas unitarias realizadas sobre la capa de lógica de negocio.

El capítulo 8 expone de las conclusiones del proyecto realizado.

Finalmente, en el capítulo 9 se presentarán los posibles trabajos futuros a realizar en el que se proporcionarán detalles concretos sobre cada una de las posibles futuras implementaciones.

2. Estado del arte

Actualmente son muchos los sistemas de gestión de vehículos existentes en el mercado, donde cada uno de ellos se adapta en alguna medida a los requisitos de los clientes finales. Estos sistemas suelen estar formados por múltiples módulos donde cada módulo tiene una funcionalidad bien definida y se encarga de distintos aspectos dentro de una empresa de alquiler de vehículos.

En este capítulo se realiza un estudio de mercado donde se presentan algunos de los sistemas de gestión alternativos ya existentes en el mercado, un análisis de la funcionalidad ofrecida por estos y las principales mejoras ofrecidas por el software desarrollado en este proyecto.

2.1 Los sistemas de gestión alternativos en la actualidad

Los sistemas de gestión de vehículos más conocidos actualmente en el mercado ofrecen a los usuarios finales tanto soluciones en local como en la nube. La principal diferencia entre estas dos es la velocidad, coste y disponibilidad del servicio, sobre todo si los datos que almacena la aplicación están en una máquina de la propia empresa de alquiler de vehículos o por el contrario están almacenados en la nube.

2.1.1 Nubea Rent Car

Una de las soluciones más modernas y novedosas actualmente ofrecidas en el mercado a nivel europeo es “Nubea Rent Car” (Ilustración 1) creado por “Diagram Software Europa SEU”.

La funcionalidad ofrecida en esta solución es la de gestionar los alquileres y vehículos. Más concretamente, dispone de tres módulos que permiten gestionar los vehículos, los alquileres y generar ciertos informes en base a las estadísticas generadas.

La principal ventaja del software desarrollado en este proyecto es que dispone de nuevas funcionalidades como por ejemplo el análisis de la necesidad de vehículos para atender a las demandas de alquileres de la semana siguiente o que la base de datos de la aplicación no tiene que estar almacenada en un servidor de terceros.

Ilustración 1: Nubea Rent Car

2.1.2 EasyRentPro Standard

"Easy Rent Pro Standard" (Ilustración 2) es otra de las soluciones conocidas que lleva varios años en el mercado americano y ha sido desarrollada por una empresa llamada "Easy Rent Pro". Este software tiene dos posibles versiones, una en local y otra en la nube.

Entre las funcionalidades más relevantes que ofrece se encuentran la de gestión de vehículos, generación de informes y gestión de alquileres.

La principal ventaja de la solución desarrollada en este proyecto es que esta dispone de funcionalidades adicionales como la posibilidad de que un cliente pueda interaccionar con este y gestionar sus reservas.

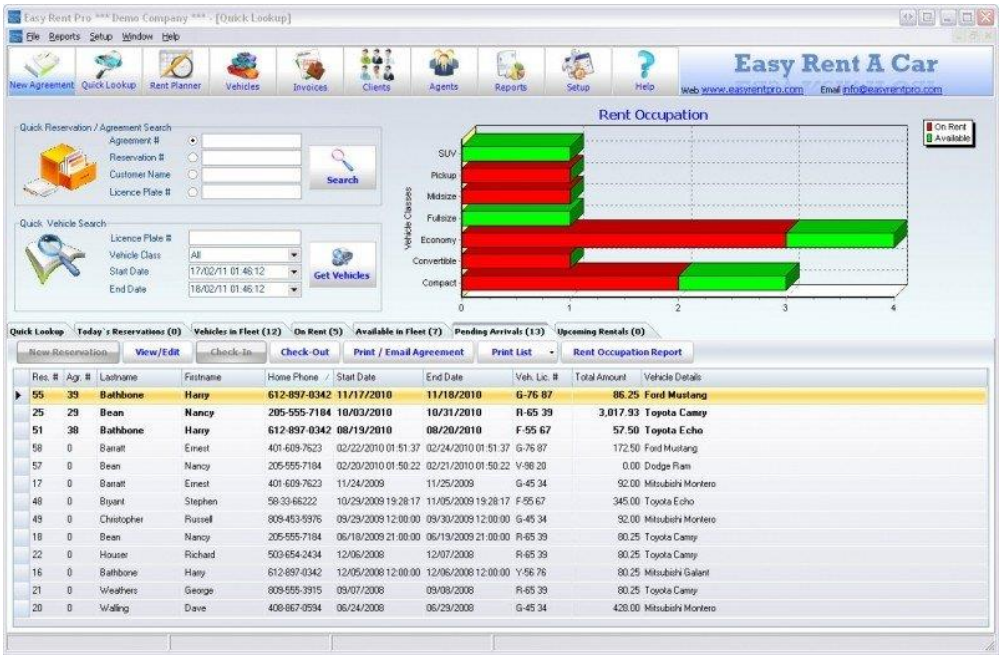

Ilustración 2: Easy Rent Pro Standard

3. Requisitos

En este capítulo se detallarán todos los requisitos de nuestra aplicación siguiendo el estándar IEEE 830-1998.

3.1 Introducción

En este apartado se van a examinar los requisitos necesarios que debe cumplir la aplicación con el fin de obtener una solución a medida. Para lograr dicho fin, se definirá el propósito de la aplicación y seguidamente se especificará el ámbito en base al entorno que la rodea y la funcionalidad que cabe esperar de esta.

3.1.1 **Propósito**

El propósito de la aplicación es ofrecer un sistema de gestión de vehículos, alquileres y reservas que puedan utilizar los empleados, administradores y clientes.

3.1.2 **Ámbito del sistema**

El sistema desarrollado recibe el nombre de “Vehicle Rental System”.
Esta aplicación permitirá realizar múltiples acciones, estando todas estas dentro del ámbito del alquiler de vehículos. Las distintas funcionalidades estarán limitadas según el tipo de usuario.

3.1.3 **Referencias**

- (1) ESPECIFICACIÓN DE REQUISITOS SEGÚN EL ESTÁNDAR IEEE830-1998.

3.2 Descripción General

La aplicación tiene como base los requisitos previamente proporcionados para la realización del proyecto. Estos requisitos simulan los proporcionados por un grupo de personas que se decidiese a montar una empresa de alquiler de vehículos y necesitasen un sistema de gestión de recursos (ERP) orientado al sector automovilístico.

3.2.1 **Perspectiva del Producto**

El sistema “Vehicle Rental System” será un sistema independiente que no dependerá de ningún otro producto software o subsistema existente para su correcto funcionamiento.

3.2.2 **Funciones del Producto**

La aplicación tiene tres tipos de usuarios que son cliente, empleado y administrador. A grandes rasgos, el sistema permitirá realizar a cada uno de ellos una serie acciones una vez se identifique y acceda dentro de la aplicación.

Los usuarios, podrán acceder al módulo de reservas y crear, buscar, editar o eliminar las reservas.

Los empleados, tendrán acceso los módulos de reservas, transferencias, alquileres e informes.
- Mediante el módulo de reservas, podrán buscar, actualizar o eliminar una reserva.
- Mediante el módulo de transferencias, podrán realizar el check-in o el check-out de una transferencia de un vehículo entre dos oficinas.

- Mediante el módulo de informes, podrán generar una serie de informes en base a la información contenida en la base de datos.
- Mediante el módulo de alquileres, podrán hacer el check-in y el check-out de los vehículos.

Por último, los administradores podrán acceder a los módulos de vehículos, clientes, extras, y oficinas y realizar las operaciones de añadir, buscar, editar o eliminar. Dentro del módulo de transferencias podrán aprobar o no las transferencias que sugiera el sistema.

3.2.3 **Características de los Usuarios**

Los usuarios de la aplicación serían los clientes, empleados y administradores de las oficinas de alquiler de vehículos. Se ha considerado que los clientes pueden tener escasos conocimientos o experiencia con las tecnologías en general, por lo que se ha realizado una interfaz lo más intuitiva posible.

En cuanto a los empleados y administradores, estos ya estarán familiarizados con la lógica y procedimientos a seguir en el mercado de alquiler de vehículos. Por ello, no debería existir ningún problema en el uso de la aplicación al ser esta bastante sencilla de usar.

3.2.4 **Restricciones**

La aplicación deberá funcionar en la versión de Windows 10.
La aplicación se realizará en el entorno de desarrollo integrado Visual Studio 2015.
Para la capa de persistencia se utilizará Entity Framework.

3.2.5 **Suposiciones y Dependencias**

La base de datos de la aplicación residirá en la máquina local en la que se esté ejecutando la aplicación.

3.3 Requisitos Específicos

Los requisitos en base a los que se ha partido para desarrollar el sistema se clasificarán en funcionales y no funcionales.

3.3.1 Requisitos funcionales

Los requisitos funcionales indican las funcionalidades debe ofrecer la aplicación a cada actor que interactúa con esta. Teniendo en cuenta las consideraciones del capítulo 5.2.1 de (2), las ilustraciones Ilustración 3, Ilustración 4, Ilustración 5 e Ilustración 6 forman el diagrama UML de casos de uso que ha sido modelado considerando cada uno de los usuarios finales que interactuarán con el sistema. Tras ello, se define cada caso de uso proporcionando más detalles sobre este.

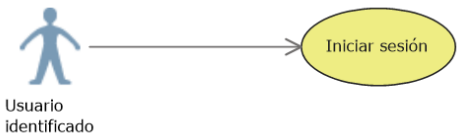

Ilustración 3: Diagrama de casos de uso para cliente, empleado y administrador.

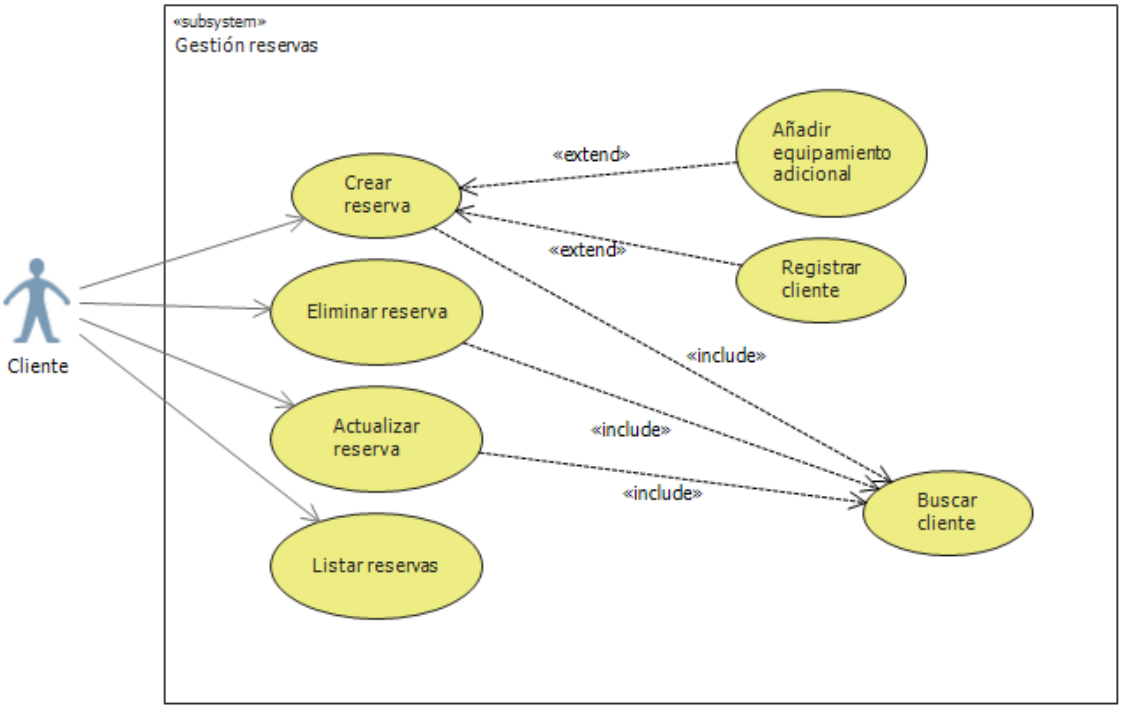

Ilustración 4: Diagrama de casos de uso para el cliente

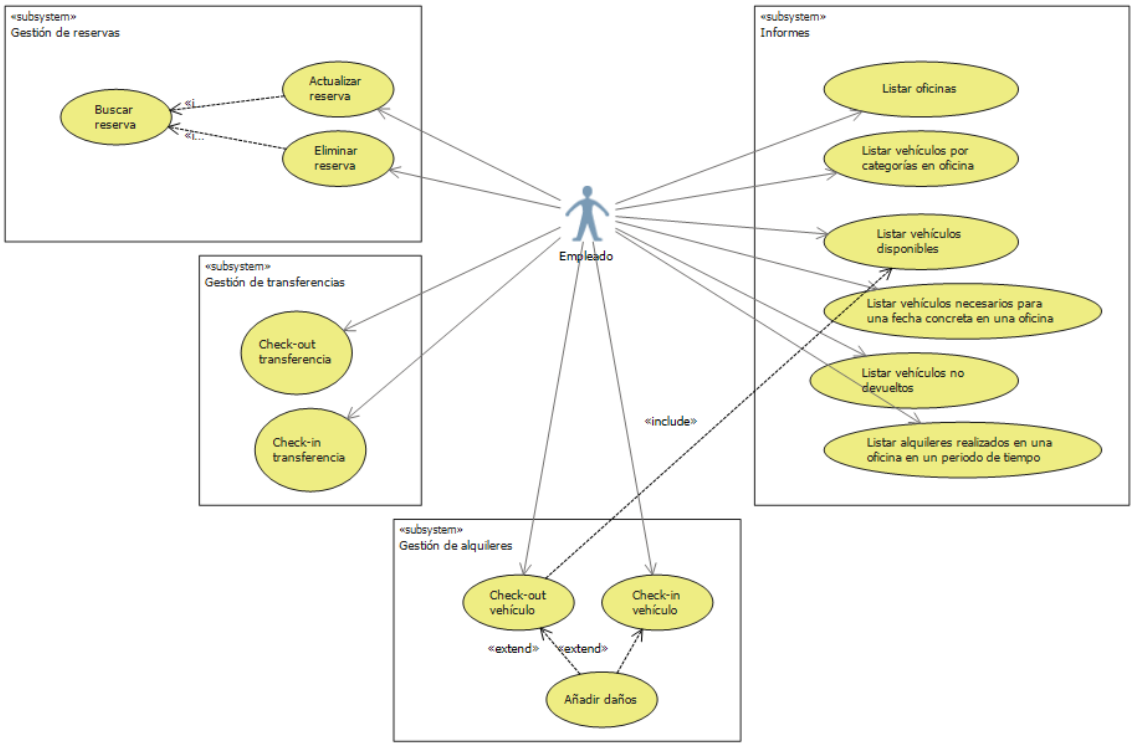

Ilustración 5: Diagrama de casos de uso para el empleado.

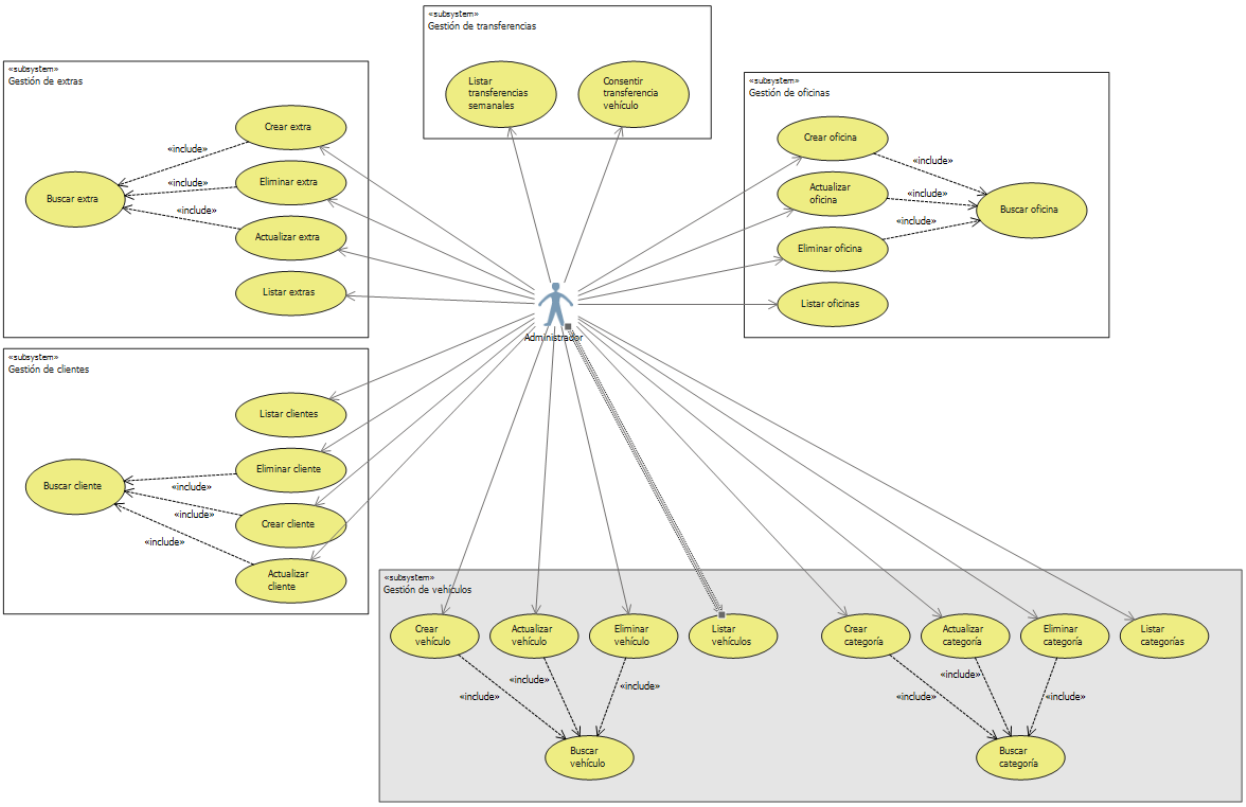

Ilustración 6: Diagrama de casos de uso para el administrador

Una vez visto el diagrama de uso desde una perspectiva global se procede a definir cada caso de uso por separado en el que se proporcionan más detalles sobre cada uno de estos.

Caso de uso 1	**Iniciar sesión**
Actor	Usuario identificado.
Descripción	Inicia sesión.
Precondiciones	Las credenciales del usuario existen en la base de datos. La cuenta no está bloqueada.
Postcondiciones	-

Caso de uso 2	Crear reserva
Actor	Cliente.
Descripción	Introduce todos los datos solicitados para poder formalizar la reserva.
Precondiciones	El cliente ha iniciado sesión. No dispone de ninguna reserva que solape con las fechas introducidas.
Postcondiciones	Si hay vehículo dentro de la categoría seleccionada, se asigna a la reserva. En caso contrario, se asigna uno de categoría superior.

Caso de uso 3	Eliminar reserva
Actor	Cliente.
Descripción	Selecciona una de sus reservas y la elimina.
Precondiciones	El cliente ha iniciado sesión. La reserva seleccionada no es una reserva antigua ni una reserva que ya haya empezado.
Postcondiciones	-

Caso de uso 4	Actualizar reserva
Actor	Cliente.
Descripción	Actualiza los datos de la reserva.
Precondiciones	El cliente ha iniciado sesión. La reserva seleccionada no es una reserva antigua ni una reserva que ya haya empezado.
Postcondiciones	Se comprueba si existe algún vehículo dentro de la categoría seleccionada. En caso contrario, se asigna uno de categoría superior.

Caso de uso 5	Listar reservas
Actor	Cliente.
Descripción	Visualiza un listado de todas sus reservas, tanto actuales como antiguas.
Precondiciones	El cliente ha iniciado sesión.
Postcondiciones	-

Caso de uso 6	Actualizar reserva
Actor	Empleado.
Descripción	Actualiza una reserva seleccionada.
Precondiciones	El empleado ha iniciado sesión. La reserva seleccionada no es una reserva antigua ni una reserva que ya haya empezado.
Postcondiciones	Se comprueba si existe algún vehículo dentro de la categoría seleccionada. En caso contrario, se asigna uno de categoría superior.

Caso de uso 7	Eliminar reserva
Actor	Empleado.
Descripción	Elimina una reserva seleccionada.
Precondiciones	El empleado ha iniciado sesión. La reserva seleccionada no es una reserva antigua ni una reserva que ya haya empezado.
Postcondiciones	-

Caso de uso 8	Check-in transferencia
Actor	Empleado.
Descripción	Introduce los datos solicitados y completa la acción.
Precondiciones	El empleado ha iniciado sesión.
Postcondiciones	-

Caso de uso 9	Check-out transferencia
Actor	Empleado.
Descripción	Introduce los datos solicitados y completa la acción.
Precondiciones	El empleado ha iniciado sesión.
Postcondiciones	-

Caso de uso 10	Check-in vehículo
Actor	Empleado.
Descripción	Introduce los datos solicitados y completa la acción.
Precondiciones	El empleado ha iniciado sesión.
Postcondiciones	-

Caso de uso 11	Check-out vehículo
Actor	Empleado.
Descripción	Introduce los datos solicitados y completa la acción.
Precondiciones	El empleado ha iniciado sesión.
Postcondiciones	-

Caso de uso 12	Listar oficinas
Actor	Empleado.
Descripción	Genera un informe con la información de todas las oficinas existentes.
Precondiciones	El empleado ha iniciado sesión.
Postcondiciones	-

Caso de uso 13	Listar vehículos por categorías en oficina
Actor	Empleado.
Descripción	Selecciona una oficina y genera un informe con todos los vehículos asignados a esta ordenados por categorías.
Precondiciones	El empleado ha iniciado sesión.
Postcondiciones	-

Caso de uso 14	Listar vehículos disponibles
Actor	Empleado.
Descripción	Selecciona una oficina y genera un informe con los vehículos disponibles en esta.
Precondiciones	El empleado ha iniciado sesión.
Postcondiciones	-

Caso de uso 15	Listar vehículos necesarios para una fecha concreta en una oficina
Actor	Empleado.
Descripción	Selecciona una oficina, una fecha y genera un informe con los vehículos necesarios para satisfacer la demanda de vehículos.
Precondiciones	El empleado ha iniciado sesión.
Postcondiciones	-

Caso de uso 16	Listar vehículos no devueltos
Actor	Empleado.
Descripción	Genera un informe con todos los vehículos no devueltos.
Precondiciones	El empleado ha iniciado sesión.
Postcondiciones	-

Caso de uso 17	Listar alquileres realizados en una oficina en un periodo de tiempo
Actor	Empleado.
Descripción	Selecciona una oficina y un periodo de tiempo y genera un informe que muestra los alquileres formalizados en esa oficina en dicho periodo de tiempo.
Precondiciones	El empleado ha iniciado sesión.
Postcondiciones	-

Caso de uso 18	Crear extra
Actor	Administrador.
Descripción	Introduce la información relativa a un extra y lo asigna a una oficina.
Precondiciones	El administrador ha iniciado sesión.
Postcondiciones	-

Caso de uso 19	Eliminar extra
Actor	Administrador.
Descripción	Si el extra no ha sido asignado a ninguna reserva lo elimina. En caso contrario, lo dejará en estado retirado conservando así toda la información del extra sin que este se pueda volver a utilizar.
Precondiciones	El administrador ha iniciado sesión.
Postcondiciones	-

Caso de uso 20	Actualizar extra
Actor	Administrador.
Descripción	Actualiza la información del extra seleccionado.
Precondiciones	El administrador ha iniciado sesión.
Postcondiciones	-

Caso de uso 21	Listar extras
Actor	Administrador.
Descripción	Visualiza un listado de todos los extras.
Precondiciones	El administrador ha iniciado sesión.
Postcondiciones	-

Caso de uso 22	Crear cliente
Actor	Administrador.
Descripción	Crea un cliente tras rellenar todos los datos relativos a este.
Precondiciones	El administrador ha iniciado sesión. El cliente tiene tarjeta de crédito. El cliente tiene permiso de conducir válido.
Postcondiciones	-

Caso de uso 23	Eliminar cliente
Actor	Administrador.
Descripción	Elimina el cliente seleccionado.
Precondiciones	El administrador ha iniciado sesión. El cliente seleccionado no tiene ninguna reserva asignada.
Postcondiciones	-

Caso de uso 24	Actualizar cliente
Actor	Administrador.
Descripción	Actualiza la información del cliente.
Precondiciones	El administrador ha iniciado sesión. La nueva información sobre el cliente es válida.
Postcondiciones	-

Caso de uso 25	Listar clientes
Actor	Administrador.
Descripción	Visualiza un listado de los clientes existentes en la base de datos.
Precondiciones	El administrador ha iniciado sesión.
Postcondiciones	-

Caso de uso 26	Crear vehículo
Actor	Administrador.
Descripción	Añade un vehículo nuevo a la base de datos.
Precondiciones	El administrador ha iniciado sesión. La información del vehículo es válida. No existe ningún vehículo con la misma matrícula. Debe existir una categoría y una oficina a la que poder asignarlo.
Postcondiciones	-

Caso de uso 27	Eliminar vehículo
Actor	Administrador.
Descripción	Elimina el vehículo seleccionado. En caso contrario, lo deja en estado retirado por lo que no se guardaría la información de este pero no se podría volver a asignar ni utilizar.
Precondiciones	El administrador ha iniciado sesión. El vehículo seleccionado no ha sido asignado a ninguna reserva o transferencia.
Postcondiciones	-

Caso de uso 28	Actualizar vehículo
Actor	Administrador.
Descripción	Actualiza los datos del vehículo.
Precondiciones	El administrador ha iniciado sesión. Los nuevos datos introducidos son correctos.
Postcondiciones	-

Caso de uso 29	Listar vehículos
Actor	Administrador.
Descripción	Visualiza un listado de todos los vehículos existentes en la base de datos.
Precondiciones	El administrador ha iniciado sesión.
Postcondiciones	-

Caso de uso 30	Crear categoría
Actor	Administrador.
Descripción	Añade una nueva categoría a la base de datos.
Precondiciones	El administrador ha iniciado sesión. La información introducida es válida.
Postcondiciones	-

Caso de uso 31	Actualizar categoría
Actor	Administrador.
Descripción	Actualiza la información de la categoría seleccionada.
Precondiciones	El administrador ha iniciado sesión. La información introducida es válida.
Postcondiciones	-

Caso de uso 32	Eliminar categoría
Actor	Administrador.
Descripción	Elimina la categoría seleccionada.
Precondiciones	El administrador ha iniciado sesión. La categoría seleccionada no tiene ningún vehículo asignado.
Postcondiciones	-

Caso de uso 33	Listar categorías
Actor	Administrador.
Descripción	Visualiza un listado de todas las categorías existentes en la base de datos.
Precondiciones	El administrador ha iniciado sesión.
Postcondiciones	-

Caso de uso 34	Crear oficina
Actor	Administrador.
Descripción	Añade una nueva oficina a la base de datos.
Precondiciones	El administrador ha iniciado sesión. No existe tal oficina en la base de datos.
Postcondiciones	-

Caso de uso 35	Eliminar oficina
Actor	Administrador.
Descripción	Elimina la oficina seleccionada o la deja como retirada conservando los datos de esta y haciendo que no se pueda volver a usar.
Precondiciones	El administrador ha iniciado sesión. No existen reservas pendientes con dicha oficina. No existen transferencias pendientes con dicha oficina. No existe ningún vehículo que no esté en estado retirado en la oficina. No existe ningún extra que no esté en estado retirado en la oficina.
Postcondiciones	-

Caso de uso 36	Actualizar oficina
Actor	Administrador.
Descripción	Actualiza los datos de la oficina en la base de datos.
Precondiciones	El administrador ha iniciado sesión. No existe ninguna oficina con la misma dirección en la base de datos.
Postcondiciones	-

Caso de uso 37	Listar oficinas
Actor	Administrador.
Descripción	Visualiza un listado con todas las oficinas existentes en la base de datos.
Precondiciones	El administrador ha iniciado sesión.
Postcondiciones	-

Caso de uso 38	Listar transferencias semanales
Actor	Administrador.
Descripción	Genera un listado de las posibles transferencias para satisfacer la demanda de reservas de la semana siguiente.
Precondiciones	El administrador ha iniciado sesión.
Postcondiciones	-

Caso de uso 39	Consentir transferencia semanal
Actor	Administrador.
Descripción	Aprueba o no la transferencia de un vehículo.
Precondiciones	El administrador ha iniciado sesión.
Postcondiciones	-

3.3.2 Requisitos no funcionales

Los requisitos no funcionales definen las limitaciones sobre las funciones que debe ofrecer el sistema.

El software contempla una serie de requisitos de calidad de la ISO 25010 descrita en su página web (3). Entre los requisitos de calidad que consideraremos más relevantes para esta solución dado su ámbito de aplicación, se encuentran limitaciones de rendimiento, usabilidad, seguridad y mantenibilidad.

Rendimiento

La aplicación deberá funcionar de forma fluida sin excesivos tiempos de carga permitiendo que el usuario pueda navegar sin complicaciones. En el caso de tener que cargar la base de datos el tiempo de carga será proporcional al tamaño de dicha base de datos.

Usabilidad

La interfaz que se presente deberá proporcionar métodos de protección ante posibles errores de usuario, de forma que los campos de entrada numéricos únicamente permitirán la entrada de valores numéricos. También se aplicarán validaciones sobre los campos de las fechas introducidas.

Adicionalmente, deberá presentar una interfaz con un diseño intuitivo y sencillo de manera que su uso no le suponga gran esfuerzo al usuario final. Por ello, el usuario deberá ser capaz de entender cómo usar el sistema tras haber tenido un primer contacto con este.

Seguridad

Los datos de identificación de personal, material y ubicaciones deberán ser guardados de manera segura.

En lo referente a las contraseñas que se guarden en la base de datos, estas deberán estar encriptadas por si llegase a existir un acceso no deseado a la máquina en la que se almacena la base de datos.

Adicionalmente, para algunas de las acciones que puedan llegar a realizar los empleados se contará con el no repudio de estas, de manera que al realizar determinada acción esta se asocie a quien la ha realizado. Entre estas se encuentran las acciones de check-in y check-out realizadas

por el empleado sobre vehículos y transferencias.

Mantenibilidad

La aplicación estará dividida en módulos, de manera que se reduzcan en la medida de lo posible el número de dependencias entre estos. Esto garantizará que la futura mantenibilidad del código se facilite en gran medida y que además se reduzcan posibles errores. Para lograr dicho fin, se utilizará una arquitectura que contará con varias capas y además el código de los distintos módulos estará agrupado en distintas secciones.

4. Análisis

Una vez establecidos los requisitos funcionales y no funcionales con los que debe contar la aplicación, se procede a la creación de un diagrama de clases inicial en el que se describe el problema.

El diagrama de clases inicial mostrado en la Ilustración 7 presenta una idea de todas las entidades con las que va a trabajar la aplicación y las relaciones existentes entre estas. Esta aproximación inicial será posteriormente mejorada considerando diversos factores como vienen siendo las tecnologías a utilizar, la optimización en el diseño de este, o la adaptación a futuros posibles cambios siendo estos algunos de todos los aspectos que se verán durante la etapa de diseño.

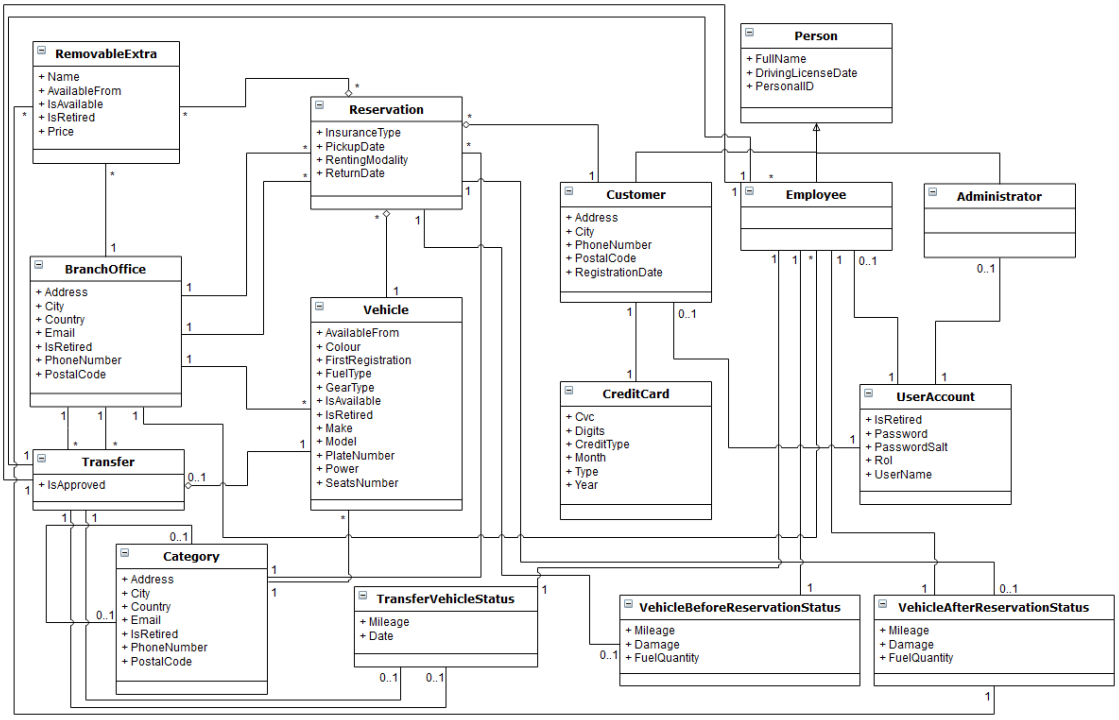

Ilustración 7: Diagrama de clases inicial

El diagrama se ha creado teniendo en cuenta las entidades presentes y las relaciones entre estas en la etapa de requisitos. Las asociaciones creadas en este diagrama inicial son todas bidireccionales, de manera que será durante la etapa de diseño donde se decida cuáles de estas pueden ser unidireccionales en base al diseño del sistema que se vaya a implementar.

5. Diseño

Una vez finalizadas las etapas de estudio de mercado, definidos los requisitos y realizado el análisis, se procederá con la fase de diseño de la aplicación. En esta fase, se decide el estilo arquitectónico que tendrá la aplicación, se proporcionan más detalles realizando a su vez mejoras sobre en diagrama de clases definido en la etapa anterior y se mostrarán las transiciones entre las distintas ventanas de la aplicación.

5.1 Decisiones en el diseño arquitectónico

El diseño arquitectónico es un proceso cuyo objetivo es diseñar una organización del sistema que cubra los requerimientos funcionales y no funcionales de la aplicación.

Durante el proceso de diseño arquitectónico, se han tomado algunas decisiones estructurales que afectan profundamente al sistema y a su futuro proceso de implementación. Tal y como se describe en el capítulo 6.1 de (2) junto con la estrecha relación que existente entre los requisitos no funcionales y a la estructura arquitectónica, la arquitectura seleccionada ha sido aquella que más beneficia la seguridad y la mantenibilidad de la aplicación. En este caso, se ha optado por una arquitectura multicapa cerrada donde el hecho de que sea cerrada implica que una solicitud va de capa en capa, sin poder saltarse ninguna capa de por medio, eliminando así las dependencias entre distintas capas y creando capas bien diferenciadas.

5.2 Diseño del diagrama de clases

Teniendo en cuenta las recomendaciones descritas en el capítulo 5.3.1 del libro (2) y en base a toda la todos los requisitos y funcionalidades indicados hasta este punto, se ha modificado el diagrama de clases definido en la etapa de análisis de manera que se tengan en cuenta:

- Las tecnologías utilizadas para implementar la solución mediante la especificación del tipado de cada atributo y añadido de los atributos ID de cada clase.
- Simplificaciones de clases por medio de la extracción de atributos comunes y la inclusión de estos en una clase padre, utilizando, por tanto, la herencia. Un claro ejemplo se puede observar en las clases referentes a los estados de las transferencias y reservas
- Simplificaciones de relaciones existentes entre clases por medio de la creación de asociaciones unidireccionales. Esto se ha realizado teniendo en cuenta que clases del modelo final no necesitan tener una referencia a otras clases, pero sí la posibilidad de ser referenciadas desde la otra clase.
- La creación de algunas clases padre por medio de la anticipación al posible cambio dado que es más que probable una futura posible ampliación de la funcionalidad ofrecida por la aplicación tras haber terminado el proyecto. Por ejemplo, se han creado las clases “Extra” y “RemovableExtra” considerando la posibilidad de que en un futuro se pueda crear una clase llamada por ejemplo “NonRemovableExtra”. La decisión de diseñar una clase “Driver” se ha tomado para no imponer la condición de que todas las personas registradas en la aplicación tuviesen un permiso de conducir, es decir, para no imponer dicha restricción a los administradores.

Como resultado del proceso del diseño en el que se han considerado los puntos mencionados anteriormente, se ha obtenido el diagrama de clases mostrado en la Ilustración 8Ilustración 8.

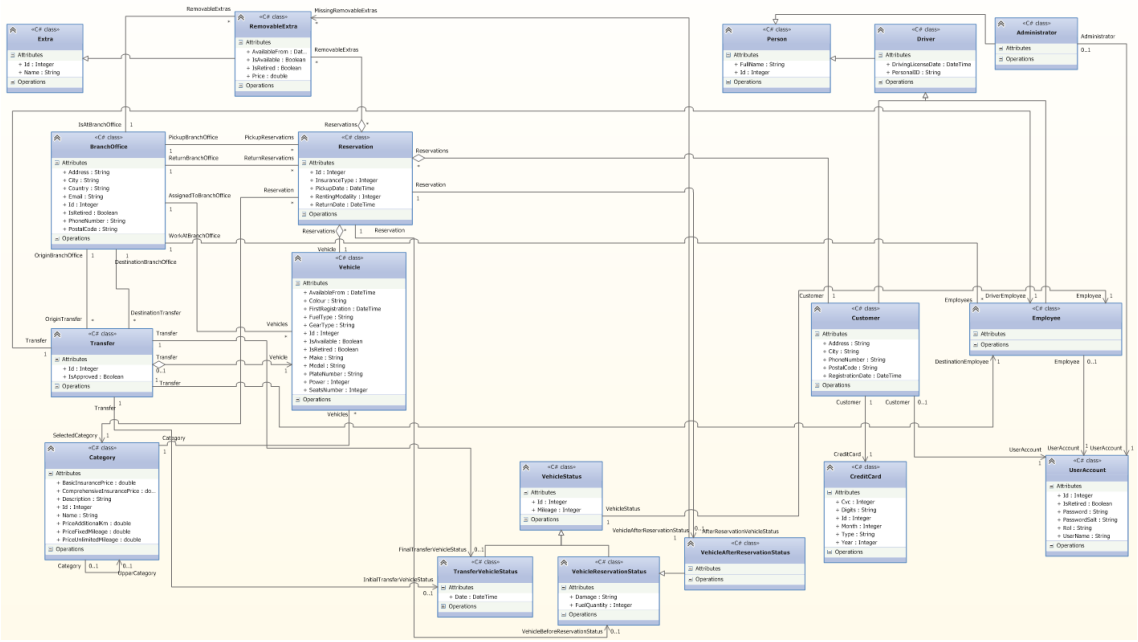

Ilustración 8:Diagrama de clases obtenido en la fase de diseño

5.3 Arquitectura del sistema

La arquitectura que se ha utilizado en esta aplicación es la arquitectura multicapa cerrada mostrada en la Ilustración 9.

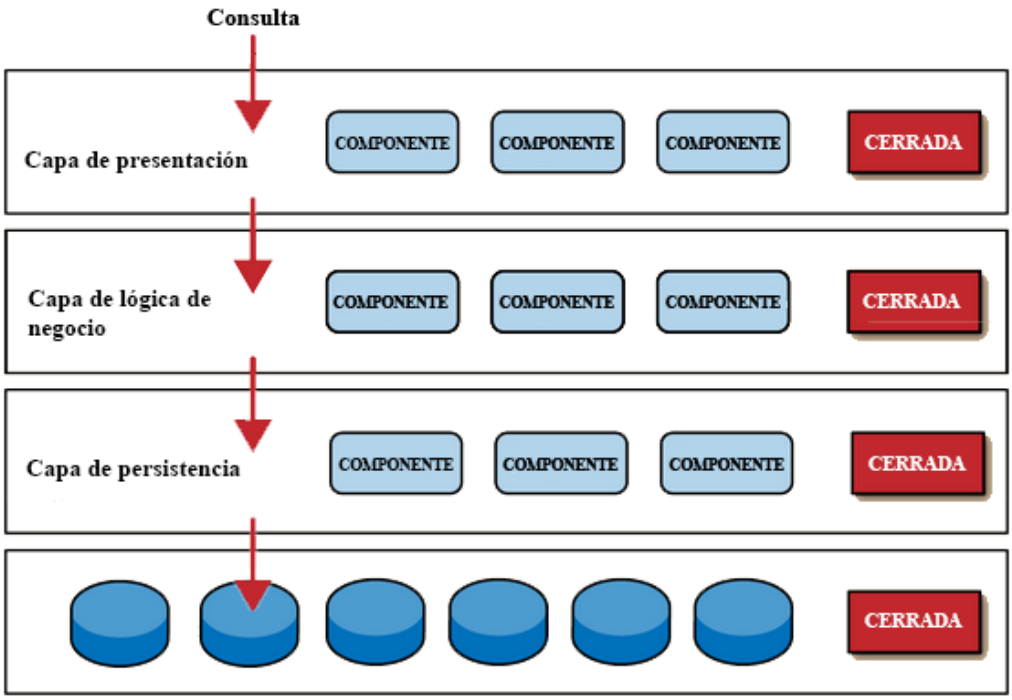

Ilustración 9: Arquitectura multicapa cerrada

- La capa de presentación: será la capa que contendrá la interfaz de usuario diseñada. Será esta la capa con la que interactúe el usuario final, de donde se obtenga la información que este introduzca y donde visualice los resultados durante la interacción. Esta capa se comunicará con la capa de lógica de negocio.
- La capa de lógica de negocio: esta capa recibe todas las peticiones solicitadas por el usuario en la capa de la presentación. Tras ello, solicita información a la capa de persistencia y aplica una serie de operaciones de lógica de negocio antes de devolver el resultado a la capa de presentación. Es decir, actúa como intermediario entre las capas de presentación y persistencia aplicando cierta lógica durante el proceso.
- La capa de persistencia: esta capa es la encargada de almacenar los datos recibidos desde la capa de lógica de negocio y también de proporcionarle los datos que la capa de lógica de negocio solicite.

Las principales ventajas de esta arquitectura son las de mantenibilidad y seguridad. Por una parte, si hiciese falta realizar cualquier modificación se podría realizar el cambio únicamente sobre la capa afectada, sin la necesidad de alterar el funcionamiento del resto de capas. Por otra parte, si hiciese falta proteger la integridad de unos datos en concreto que afectasen por ejemplo a dos de las clases con las que va a trabajar la aplicación, se podrían guardar estas en otra base de datos con un mayor índice de mecanismos de seguridad sin tener que realizar ningún cambio sobre las capas de presentación o lógica de negocio.

5.3.1 Capa de presentación

En esta capa se definen los elementos de la interfaz gráfica de usuario. Por su parte, el usuario interactúa con la aplicación a través de las distintas pantallas creadas para dicho fin. La interacción del usuario con las pantallas consiste en la introducción y la obtención de datos resultantes tras haber aplicado algún tratamiento sobre estos.

La estructura que conforma esta capa se puede observar a continuación:

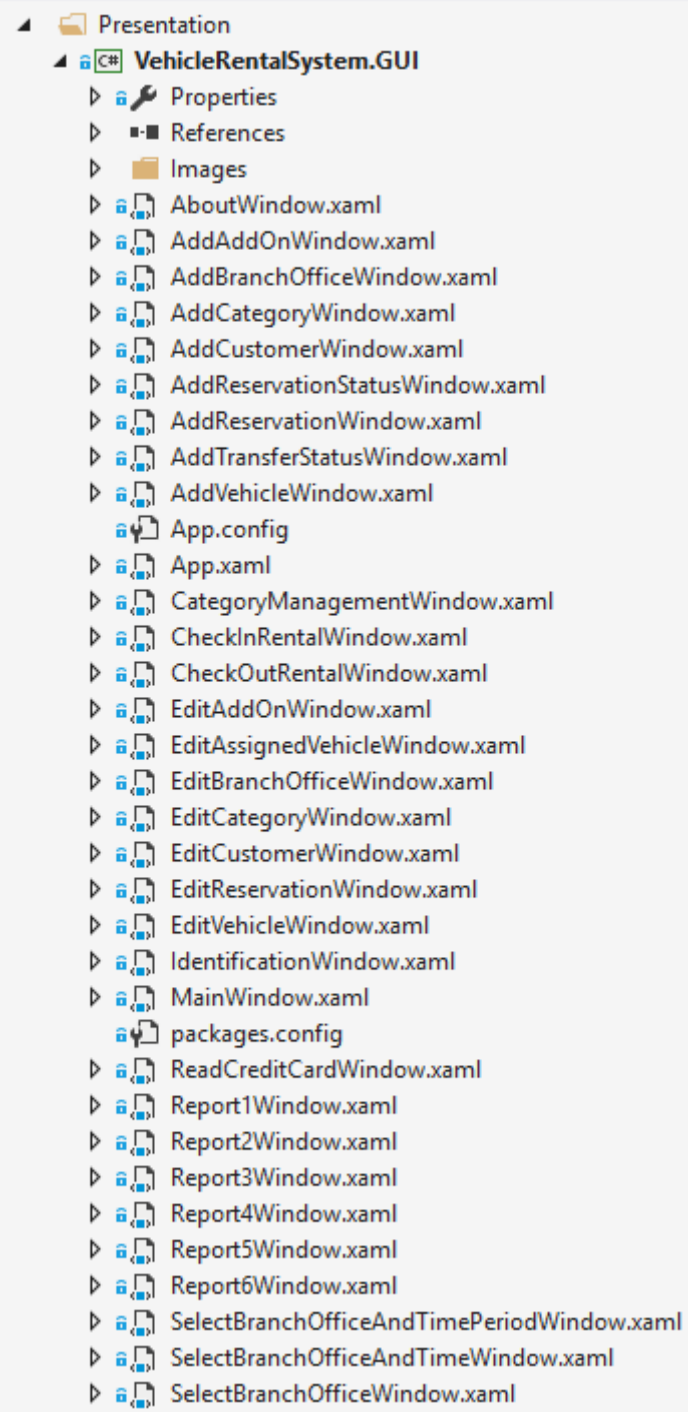

Ilustración 10: La capa de presentación

Como se puede observar, son múltiples las pantallas definidas para interactuar con el sistema. A continuación, se mostrarán dichas pantallas desde el punto de vista del diseño sin entrar en detalle en las funcionalidades disponibles para cada tipo de usuario, dado que estas ya se han definido en los requisitos funcionales de la aplicación.

Inicio de sesión

Ilustración 11: Pantalla de inicio de sesión

Al ejecutar la aplicación aparece la ventana de identificación indicada en la Ilustración 11. Cuando un usuario introduce sus credenciales, la contraseña se encripta utilizando un salteado específico para cada usuario que viene indicado en la base de datos. Tras ello se compara si la cadena de texto encriptada y salteada coincide con la existente en la base de datos. En dicho caso, el usuario accedería a la ventana principal de la aplicación mostrada en la Ilustración 12.

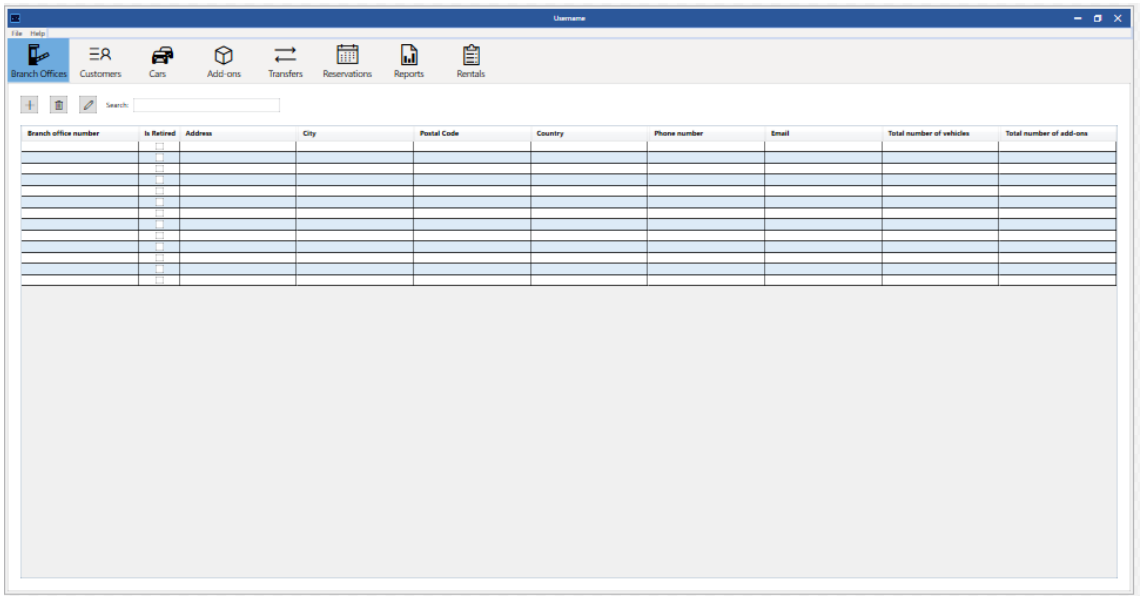

Ilustración 12: Pantalla de la ventana principal

En la ventana principal mostrada en la Ilustración 12 se puede visualizar una clara distinción entre los componentes que conforman la aplicación. Estos son los que aparecen en el menú horizontal de iconos en la parte superior de la pantalla.

Durante el diseño de la aplicación, se ha seguido el criterio de que un usuario debe visualizar únicamente aquella información que le concierne. Por ello, en función del tipo de usuario que inicie sesión, este visualizará únicamente aquellos elementos que estén relacionados con la funcionalidad que este requiera en cada momento.

Módulo de gestión de oficinas

En la Ilustración 12 aparece seleccionado dicho módulo al que tiene acceso únicamente el administrador. En la tabla se cargarán todas las oficinas existentes y las columnas mostrarán todas las características de cada oficina. Las acciones disponibles sobre una oficina previamente seleccionada en la tabla son las de borrar y editar (Ilustración 13). También existe la posibilidad de introducir una nueva oficina (Ilustración 14). Adicionalmente, se permitirá realizar la búsqueda de una oficina por medio de la introducción del identificador de la oficina.

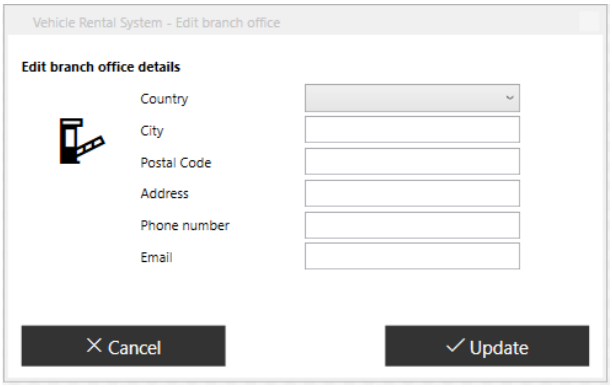

Ilustración 13: Pantalla para añadir una oficina

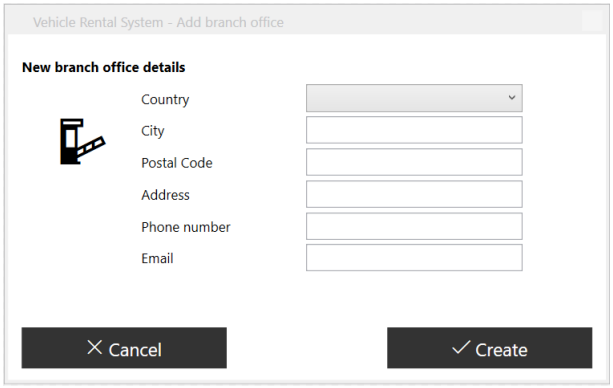

Ilustración 14: Pantalla de edición de oficina

Módulo de gestión de clientes

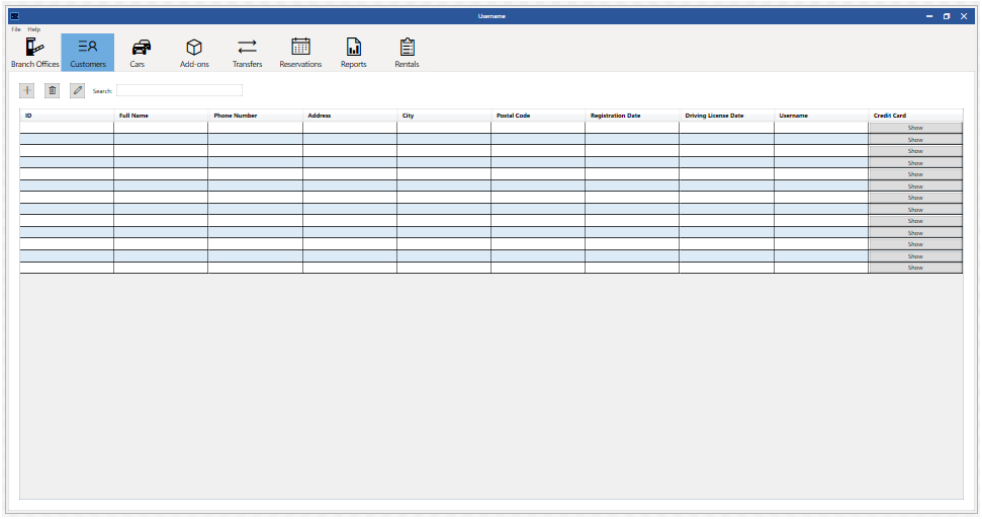

Ilustración 15: Pantalla del módulo de gestión de clientes

En la Ilustración 15 aparece seleccionado dicho módulo al que tiene acceso únicamente el administrador. En la tabla se cargarán todas las oficinas existentes y las columnas mostrarán todas las características de cada cliente. La última columna ofrece la posibilidad de visualizar los datos de la tarjeta de crédito (Ilustración 16).

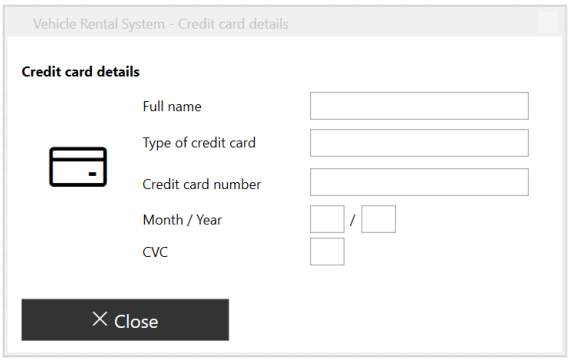

Ilustración 16: Pantalla informativa de los datos de la tarjeta de crédito

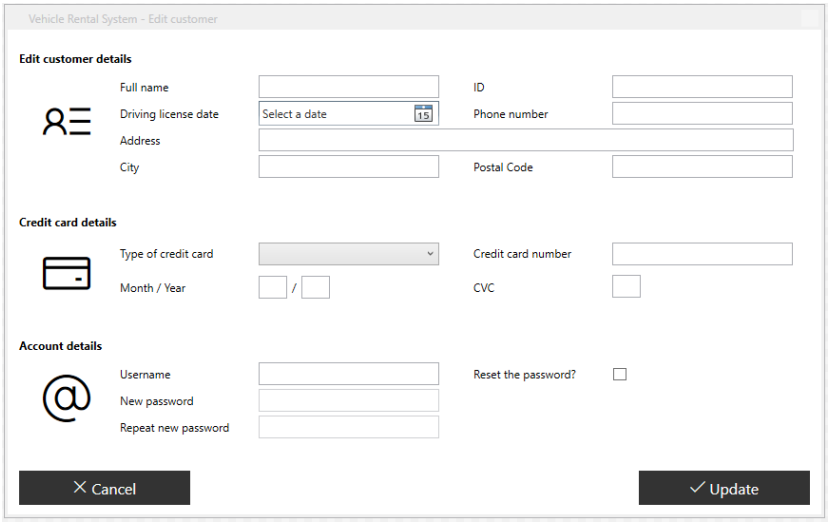

Ilustración 17: Pantalla de edición de un cliente

Las acciones disponibles sobre un cliente previamente seleccionado en la tabla son las de borrar y editar (Ilustración 17).

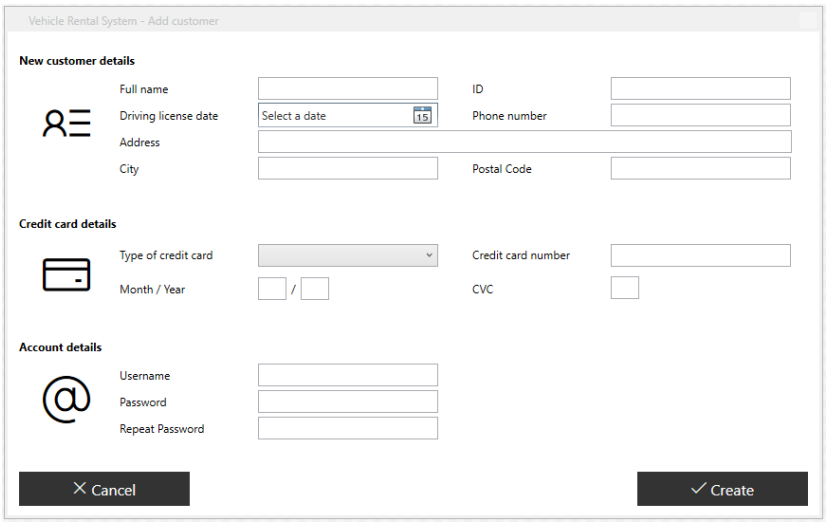

Ilustración 18: Pantalla para añadir un nuevo cliente

También existe la opción de añadir un nuevo cliente. (Ilustración 18).

Módulo de gestión de vehículos

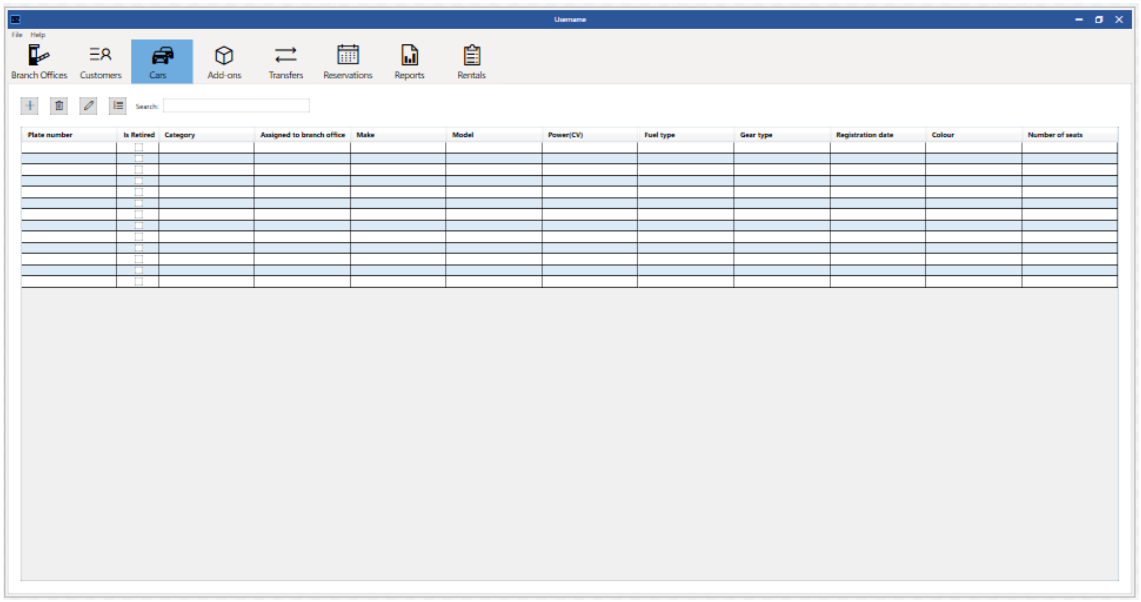

Ilustración 19: Pantalla del módulo de gestión de vehículos

Este módulo es únicamente accesible por el administrador.

En la tabla se cargan todos los vehículos existentes y las columnas muestran toda la información relativa a cada uno de estos. Las acciones disponibles sobre un vehículo previamente seleccionado en la tabla son las de borrar y editar (Ilustración 20).

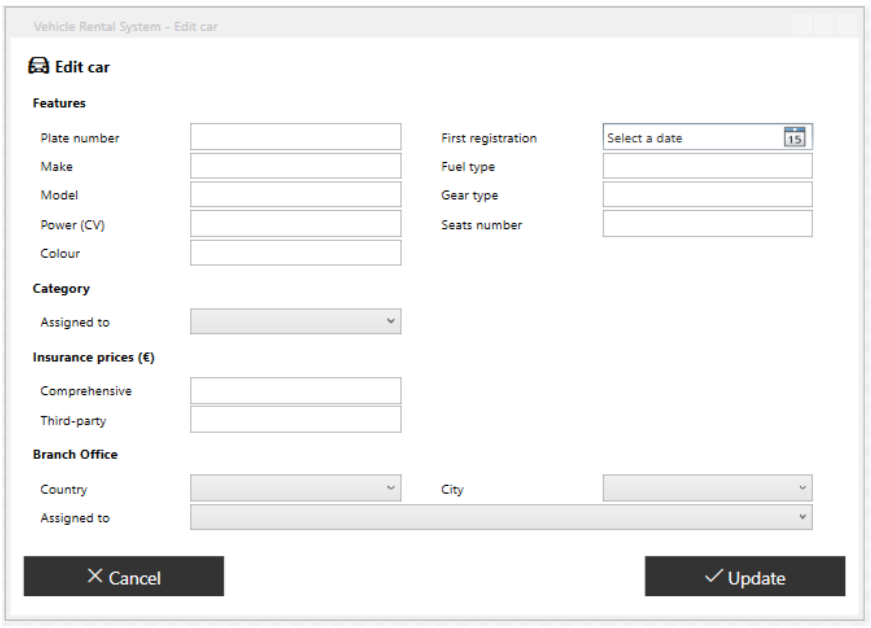

Ilustración 20: Pantalla de edición de un vehículo

Se permitirá también la posibilidad de añadir un nuevo vehículo tras insertar todos los detalles sobre este (Ilustración 21).

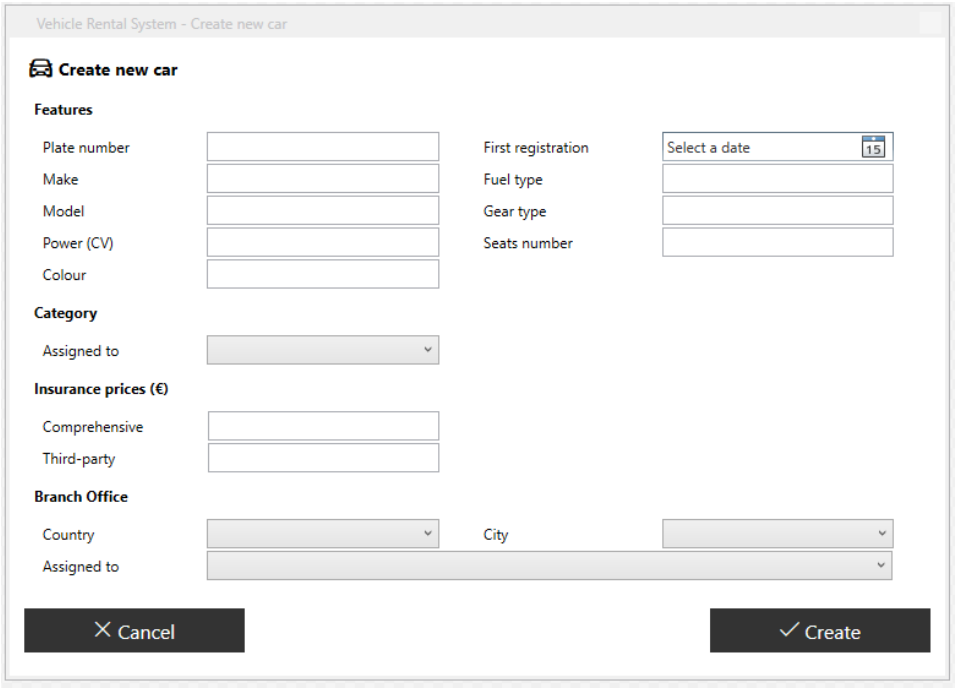

Ilustración 21: Pantalla para añadir un vehículo

En este módulo dispone además de la opción de gestionar las categorías de los vehículos (Ilustración 22).

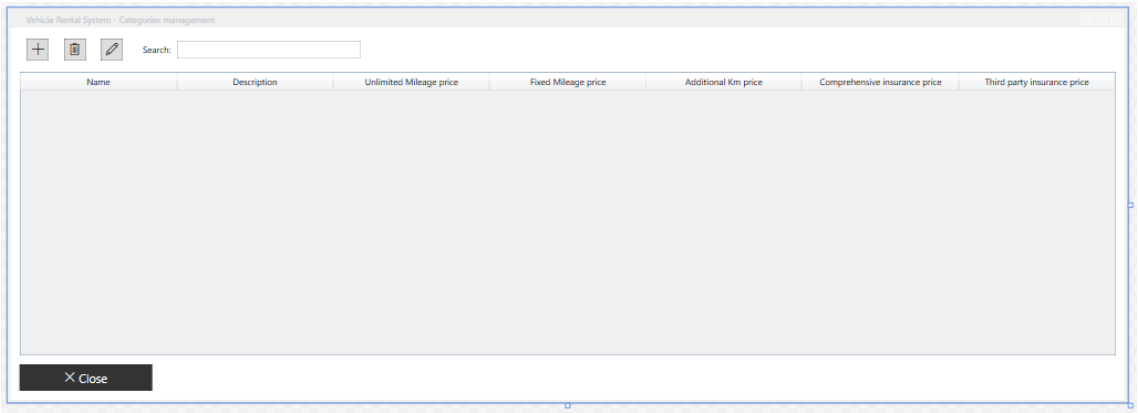

Ilustración 22: Pantalla de gestión de categorías

La pantalla de gestión de categorías (Ilustración 22) permite crear (Ilustración 25), editar (Ilustración 26) y eliminar categorías al administrador. Las categorías aparecen ordenadas en orden creciente. De esta manera, la categoría que aparezca con el identificador uno en la tabla será la categoría más alta, mientras que un valor inferior significa que es una categoría más baja. Al borrar una categoría, las categorías inferiores a esta pasarían a ser las inferiores de la categoría inmediatamente superior a esta.

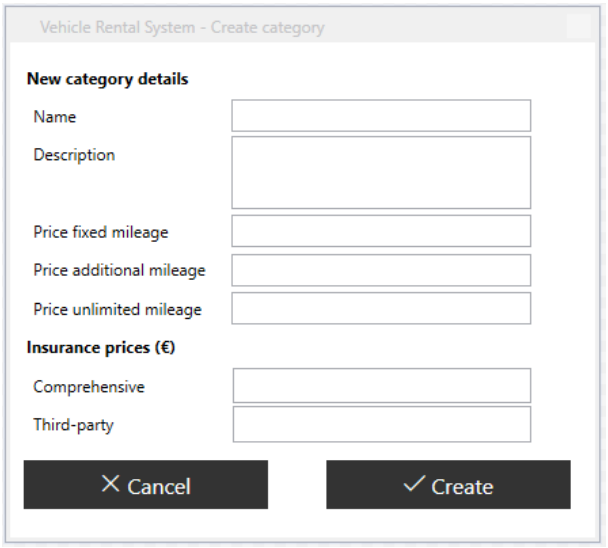

Ilustración 23: Pantalla para añadir una nueva categoría

La pantalla verificará los tipos de datos introducidos y la longitud de estos. En caso de no ser válidos, no permitirá la introducción de estos en los campos.

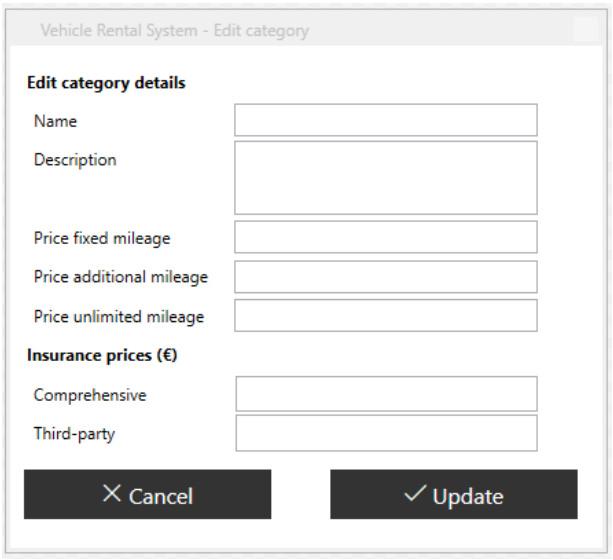

Ilustración 24: Pantalla de edición de una categoría

Tras introducir los datos solicitados y válidos en la pantalla anterior, se actualizaría la categoría en la base de datos.

Módulo de gestión de extras

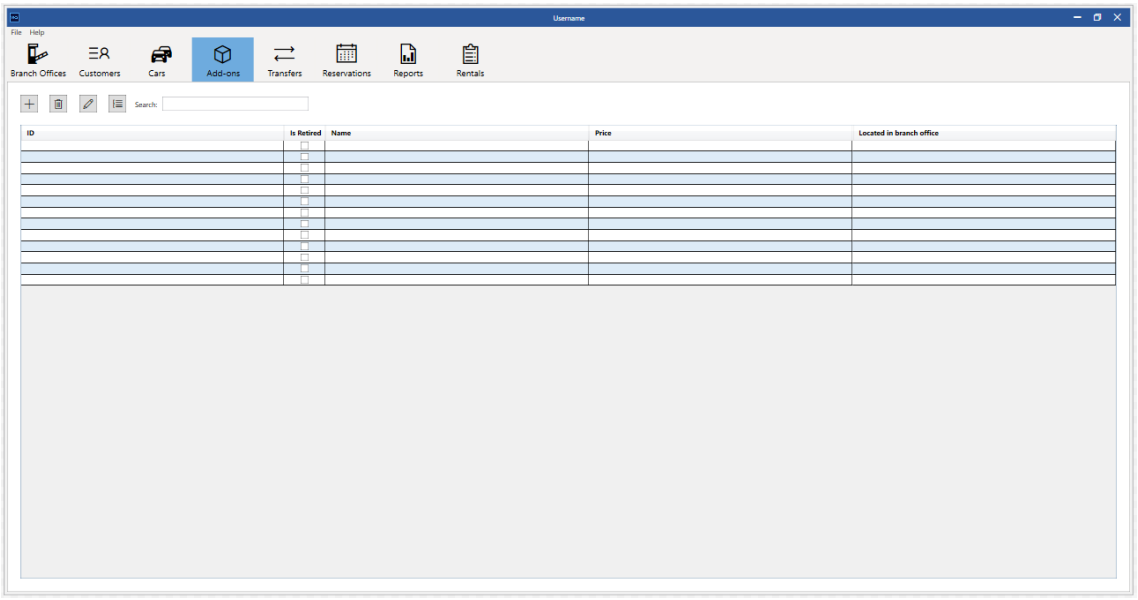

Ilustración 25: Pantalla del módulo de gestión de extras

Este módulo es únicamente accesible para el administrador y permite gestionar todos los extras existentes que están asignados a alguna oficina. Las opciones que ofrece sobre un extra seleccionado son las de borrar o editar (Ilustración 26) el extra.

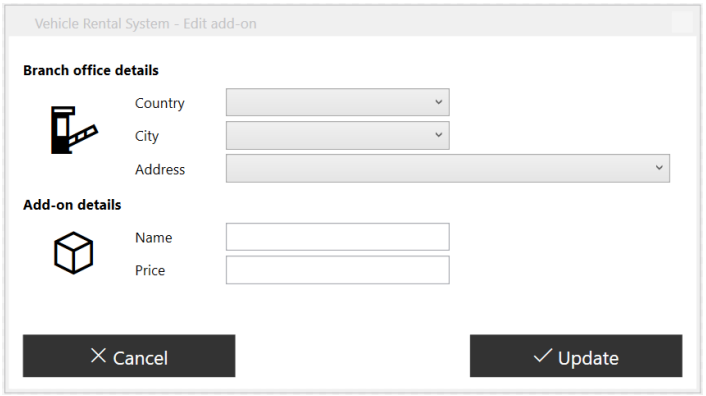

Ilustración 26: Pantalla de edición de un extra

También permite añadir un nuevo extra y asignarlo a una oficina (Ilustración 27).

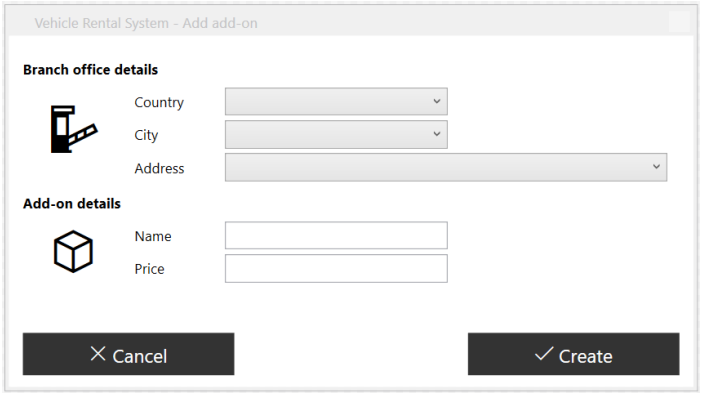

Ilustración 27: Pantalla para añadir un extra

Por último, el administrador también dispone de la opción de agrupar extras con la que se agruparán los extras por la oficina a la que estos estén asignados.

Módulo de gestión de transferencias

Este módulo (Ilustración 28) es accesible tanto por administradores como por empleados, pero las opciones que visualiza cada uno de ellos tras acceder a este son distintas.

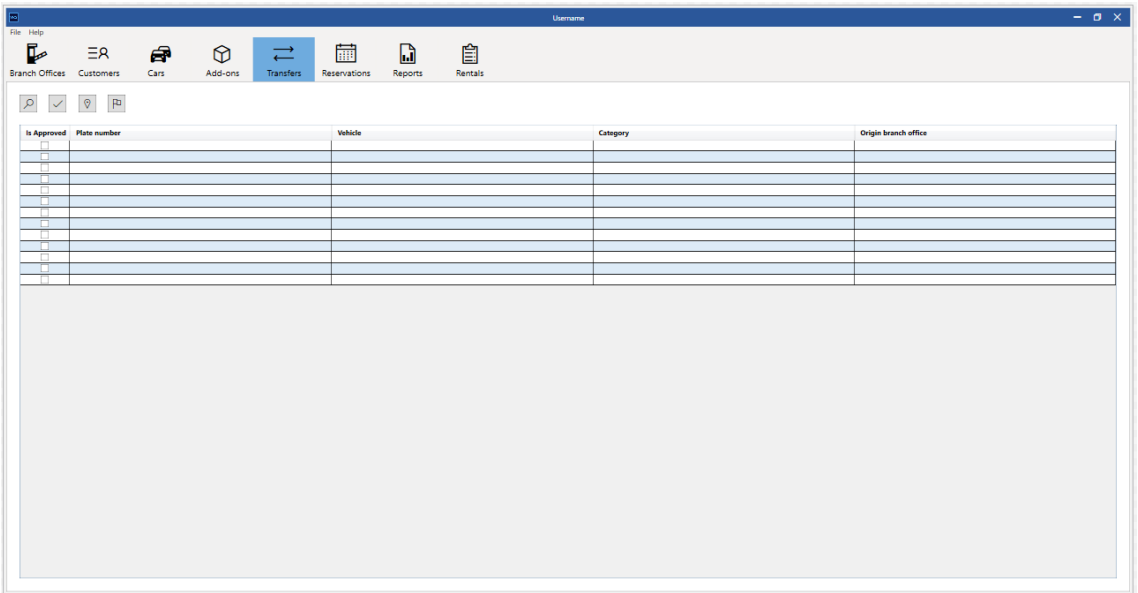

Ilustración 28: Módulo de gestión de transferencias

Tanto los administradores como los empleados visualizan el listado de transferencias. La principal diferencia es que tras darle al botón de búsqueda, los administradores visualizan todas aquellas transferencias que son posibles de realizar para la próxima semana. Una vez listadas estas, puede aprobar las transferencias si lo desea. Por su parte, el empleado visualiza el listado de transferencias aprobadas previamente por el administrador. Las acciones que puede realizar son las de hacer check-in o check-out tras realizar una selección previa de alguna transferencia.

Módulo de gestión de reservas

Este módulo es accesible tanto por los clientes como por los empleados pero las acciones que pueden realizar cada uno de ellos difieren. El usuario puede crear (Ilustración 30) , borrar o editar una reserva (Ilustración 31), mientras que el empleado solamente podrá actualizar o borrar una reserva. También pueden realizar una búsqueda de una reserva introduciendo el identificador de esta.

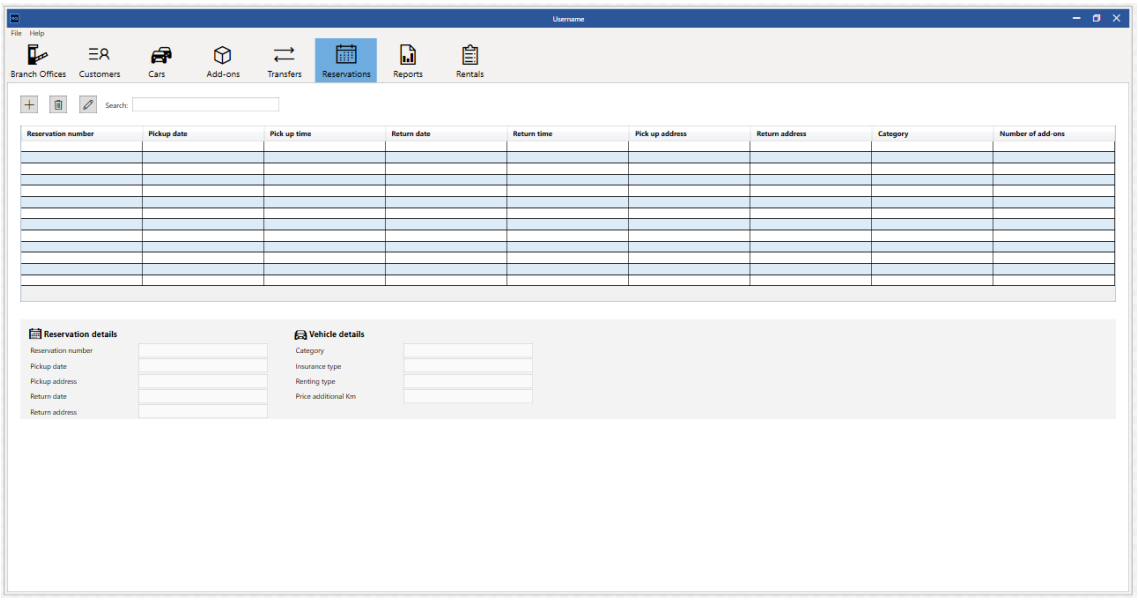

Ilustración 29: Módulo de gestión de reservas

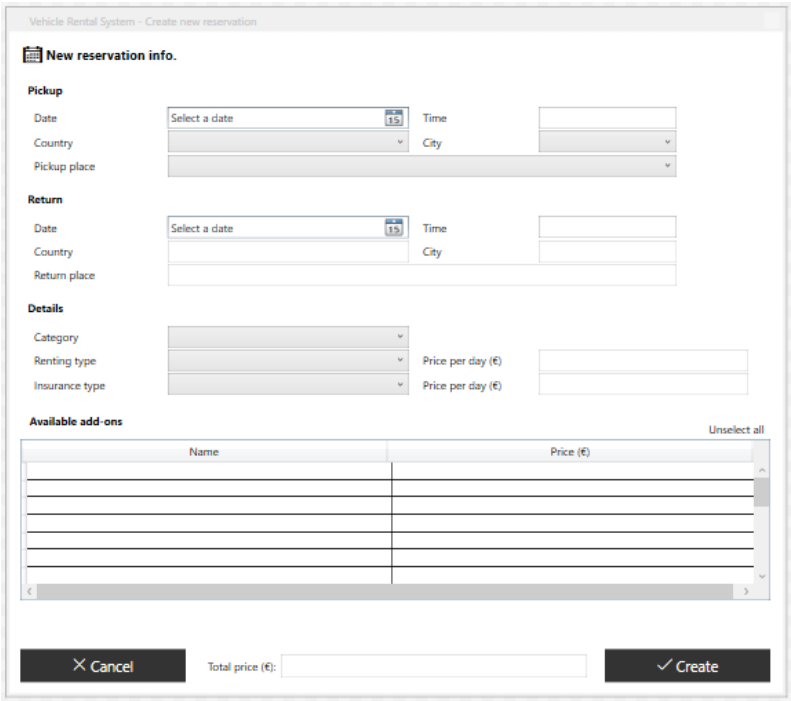

Ilustración 30: Pantalla para añadir una reserva

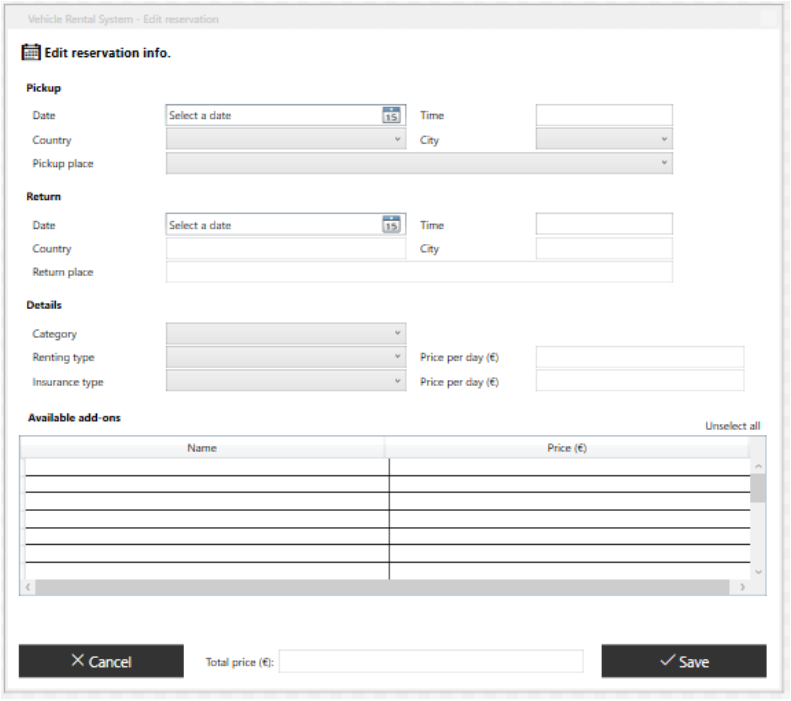

Ilustración 31: Pantalla de edición de una reserva

Módulo de gestión de informes

El empleado dispone de la opción de crear informes en base a la información contenida en la base de datos. Los distintos informes que se pueden generar se pueden observar en la Ilustración 32.

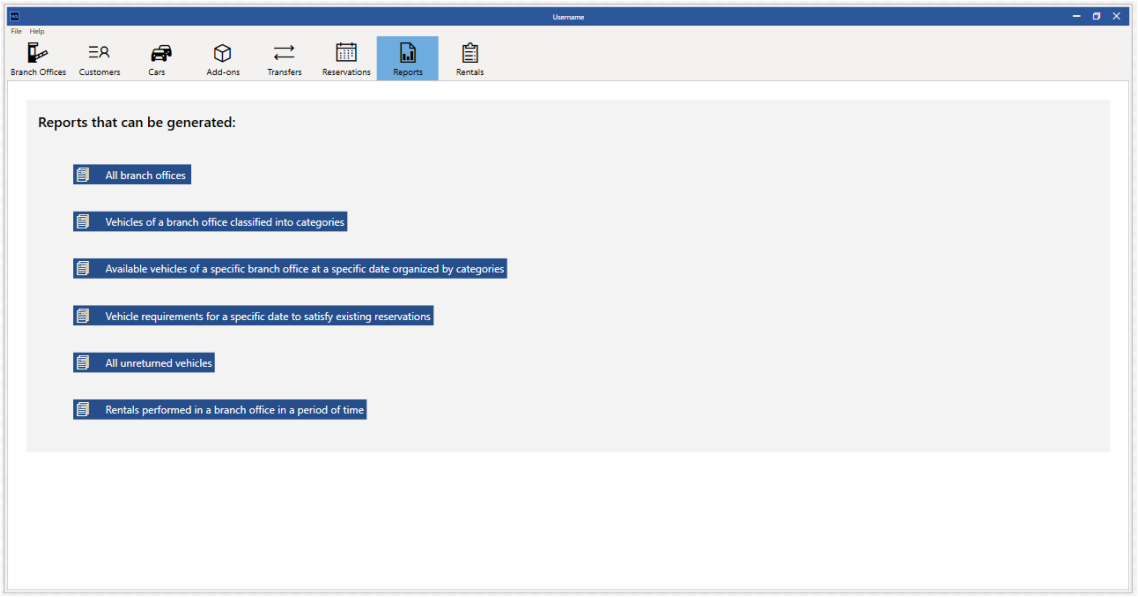

Ilustración 32: Módulo de generación de informes

Tras seleccionar cualquier tipo de informe, es posible que la aplicación solicite algún parámetro en base al cual va a generar el informe. Entre estos se puede encontrar la dirección de una oficina, una fecha o un periodo de tiempo. Tras ello, se generará el informe seleccionado y se mostrará en una ventana de previsualización. La ventana de previsualización permite visualizar, guardar en formato PDF o imprimir el informe. Dos claros ejemplos de informes generados son los de las Ilustraciones 32 y 33.

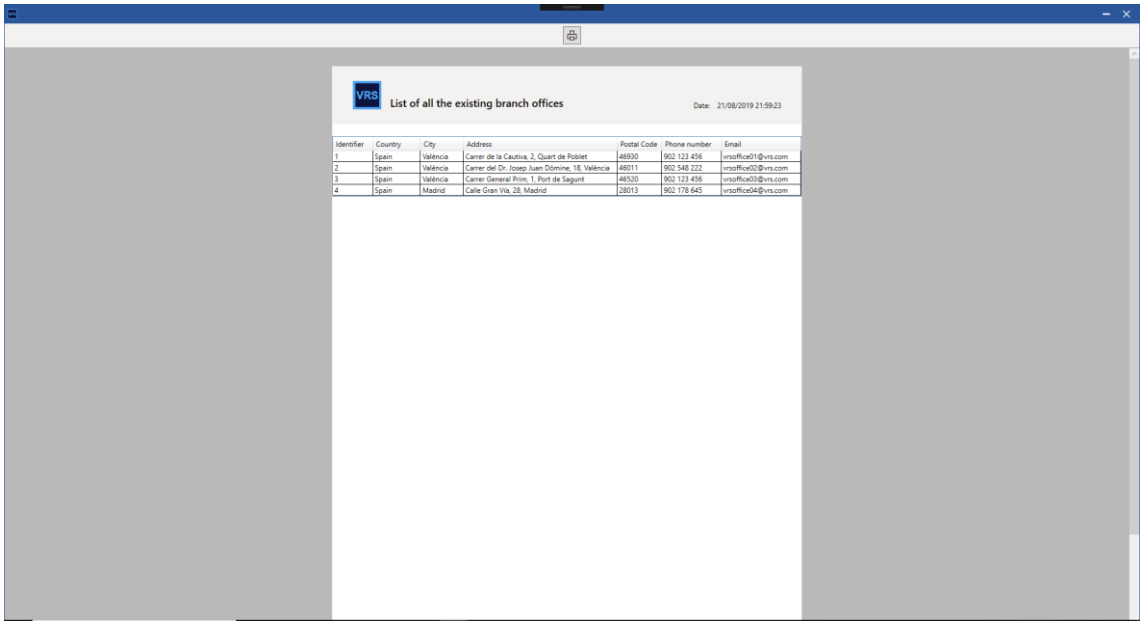

Ilustración 33: Previsualización de un informe de listado de oficinas

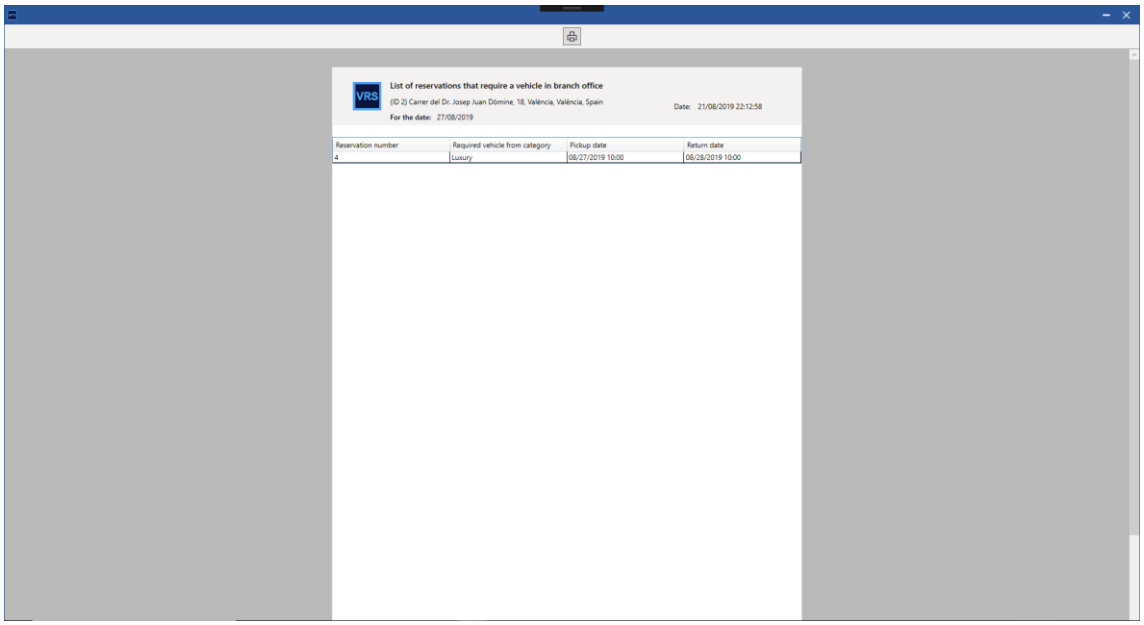

Ilustración 34: Previsualización de un informe de vehículos requeridos para una fecha en una oficina

5.3.2 Capa de lógica de negocio

Esta capa interactúa con la capa de presentación y con la capa de persistencia. La capa contiene una serie de clases definidas de forma parcial que complementan a las creadas también de forma parcial en la capa de persistencia. Estas clases se han definido gracias al diagrama de clases mostrado durante la etapa de requisitos, al que posteriormente se le realizaron pequeños ajustes durante la fase de diseño.

La lógica de esta capa hace referencia a la lógica propia de la aplicación, es decir, aplica una serie de tratamientos sobre los datos introducidos por el usuario con el fin de obtener el resultado esperado. Este resultado se podrá almacenar en la base de datos o ser mostrado por pantalla gracias a la capa de presentación. Por ello, se considera que esta es la capa que da forma a la información obtenida de la capa de persistencia.

5.3.3 Capa de persistencia

La función de esta capa es la de escribir y la de leer toda la información en la base de datos. Esta capa interactúa con la capa de lógica de negocio.

En la aplicación, esta capa la componen las siguientes clases:

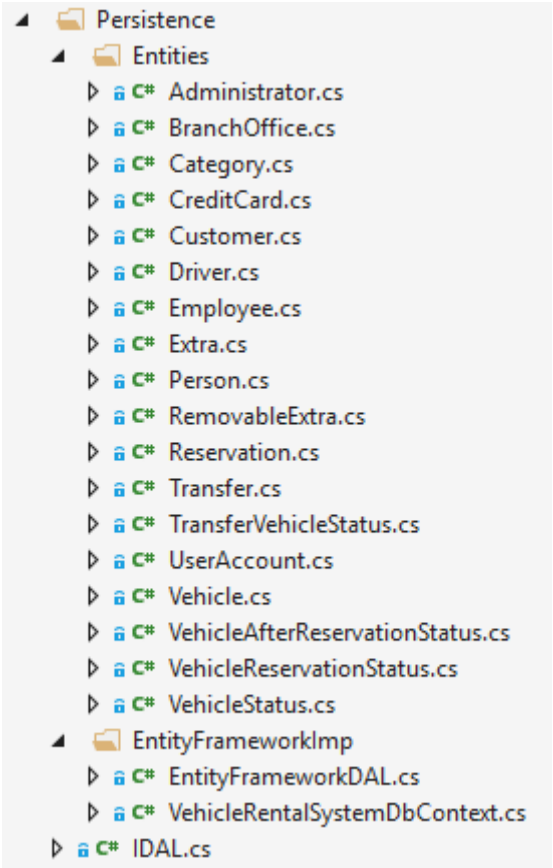

Ilustración 35: Capa de persistencia

En la Ilustración 35 se pueden observar las clases extraídas del diagrama de clases de la fase de diseño. Estas clases son definiciones parciales que contienen únicamente atributos.

En esta capa se ha trabajado con la herramienta Entity Framework. Esta herramienta es un sistema de mapeo objeto-relacional que permite que la aplicación trabaje con entidades que son objetos mientras que Entity Framework se encarga de gestionar de forma transparente la persistencia de estos objetos en la base de datos relacional.

Esta gestión transparente por parte de la herramienta se ha logrado utilizando los patrones de "Repository" y "Unit of Work" que serán explicados en más detalle en la etapa de implementación

Entity Framework permite especificar la estrategia a seguir durante la creación del modelo de base de datos. Considerando que la aplicación desarrollada es una aplicación de gestión en la cual se lanzan multitud de consultas sobre la base de datos esperando que estas sean lo más rápidas posibles, se ha optado por utilizar la mejor estrategia para conseguir un alto rendimiento a la hora de lanzar consultas sobre esta. La estrategia utilizada es la TPH (tabla por jerarquía). La principal razón de que esta sea la más eficiente es que al existir una única tabla por jerarquía toda la información se encuentra almacenada en esta evitando la necesidad de usar operadores JOIN y UNION en las consultas. Esta implementación se ha realizado considerando la descripción de dicho patrón mostrada en la web (4).

Por último, también se aplica el patrón singleton a la hora de instanciar la clase "EntityFrameworkDAL" en la capa de lógica de negocio, lo que implica que durante la ejecución de dicha clase existirá una única instancia del objeto. Esta clase implementa la estructura del patrón de acceso a datos (DAO) por medio de la implementación de una serie de métodos genéricos definidos en la interfaz IDAL que permiten realizar distintas acciones de lectura, escritura y consulta sobre los objetos almacenados en la base de datos. Esto lo hace mediante la encapsulación de la lógica necesaria para copiar los valores de los datos desde las clases del dominio del problema que está definido en la capa de lógica y la persistencia. Adicionalmente, permite realizar el guardado de los objetos en la base de datos.

6. Implementación

Tras haber finalizado las fases previas de requisitos, análisis y diseño quedan claramente definidos los requisitos funcionales y no funcionales de la aplicación además de los aspectos de diseño vistos anteriormente. Con ello, se procede a la fase de implementación de la aplicación.

Durante la fase de implementación se presentarán el entorno de desarrollo y otras herramientas utilizadas para el desarrollo de la aplicación. Posteriormente, dada la gran magnitud de líneas de código existentes, se proporcionará una explicación de la lógica seguida para implementar los métodos existentes viendo en más detalle los métodos más destacados que están implementados en la capa de lógica de negocio.

6.1 Tecnologías utilizadas

Para el desarrollo de la aplicación de escritorio para Windows se ha utilizado el entorno de desarrollo "Visual Studio 2015" de Microsoft. La versión empleada ha sido la "Ultimate" bajo una licencia de estudiante que permite el desarrollo de software con fines académicos.

Visual Studio también incorpora el explorador de servidores que permite asociar un servidor con el fin de visualizar e incluso editar el contenido de una base de datos. Por ello, permite previsualizar el contenido de cada una de las tablas contenidas en la base de datos.

Cabe destacar también el uso de Entity Framework dentro de Visual Studio. Toda la información sobre cómo utilizar dicha tecnología se ha encontrado en la web oficial indicada en la referencia (5). Esta herramienta está formada por un conjunto de tecnologías de la conocida ADO.NET que posibilitan el desarrollo de aplicaciones que trabajen con objetos y que estos objetos se guarden en la base de datos relacional de forma transparente.

Dentro de la capa de presentación se ha utilizado otra tecnología de Microsoft conocida como Windows Presentation Foundation o por sus siglas WPF. Esta herramienta incorporada también dentro de Visual Studio ofrece una infraestructura que permite el desarrollo de aplicaciones visualmente atractivas mediante un lenguaje basado en XML conocido como "eXtensible Application Markup Language" o por sus siglas XAML.

Por último, el lenguaje de programación utilizado ha sido C#, cuyo creador es Microsoft. Este lenguaje está orientado a objetos y su sintaxis básica ha derivado de C/C++ utilizando el modelo de objetos de la plataforma .NET, que es similar al de Java. Por tanto, C# es un lenguaje de programación independiente diseñado para generar aplicaciones sobre dicha plataforma.

6.2 Estructura de la solución

Durante la fase de implementación, la estructura que se ha seguido es la que se ha explicado en la fase de diseño, implementando una arquitectura dividida en tres capas.

En terminología de Visual Studio, un proyecto es conocido como una solución. Esta solución estará formada por lo que en Visual Studio se conoce como múltiples proyectos. La Ilustración 36 muestra la solución desarrollada y los proyectos que la componen.

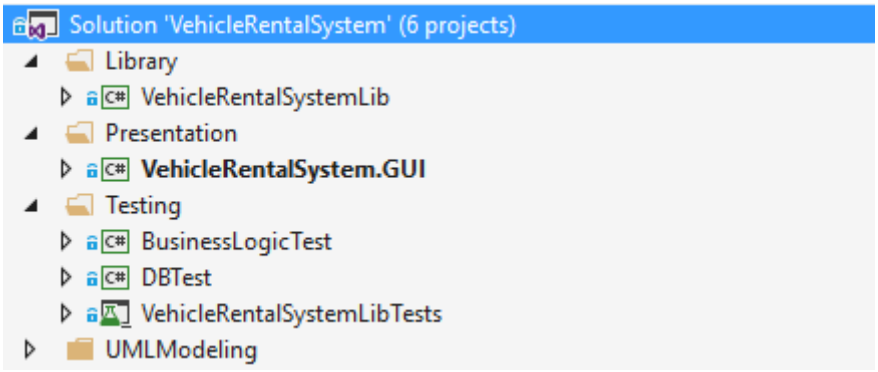

Ilustración 36: Solución implementada en Visual Studio

Dentro del proyecto de librería mostrado en la Ilustración 36, se encuentran las capas de lógica de negocio y de persistencia.

La lógica de negocio está compuesta por las carpetas de entidades y servicios. La carpeta de entidades contiene una implementación parcial de las clases utilizadas en la aplicación. Estas

clases parciales contienen la implementación de los constructores de cada clase. La carpeta de servicios contiene la implementación de la lógica de los servicios ofrecidos por la interfaz "IVehicleRentalSystem". Los servicios ofrecidos por esta interfaz se utilizarán por medio de llamadas a métodos desde la capa de presentación.

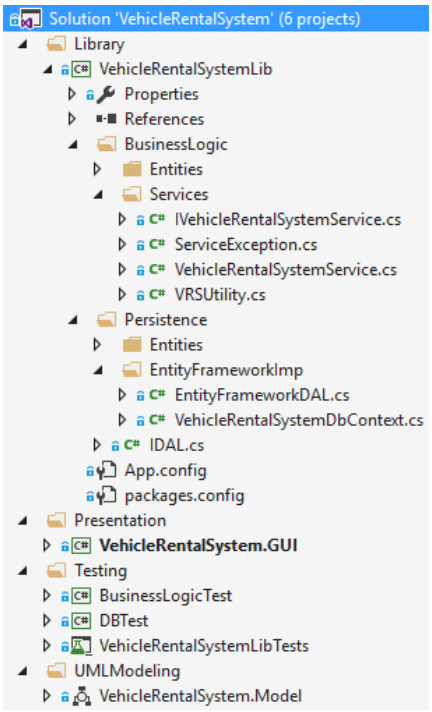

Ilustración 37: Proyecto contenedor de la librería de la aplicación

La estructura y los ficheros que presenta la capa de persistencia se explicarán en mayor detalle en el punto 6.3.3 Capa de persistencia.

La carpeta de modelado UML tiene en su interior un proyecto de modelado que contiene los diagramas de casos de uso vistos en la fase de requisitos y el diagrama de clases visto en la fase de diseño. Estos se han podido crear gracias a las herramientas ofrecidas por Visual Studio para dicho fin.

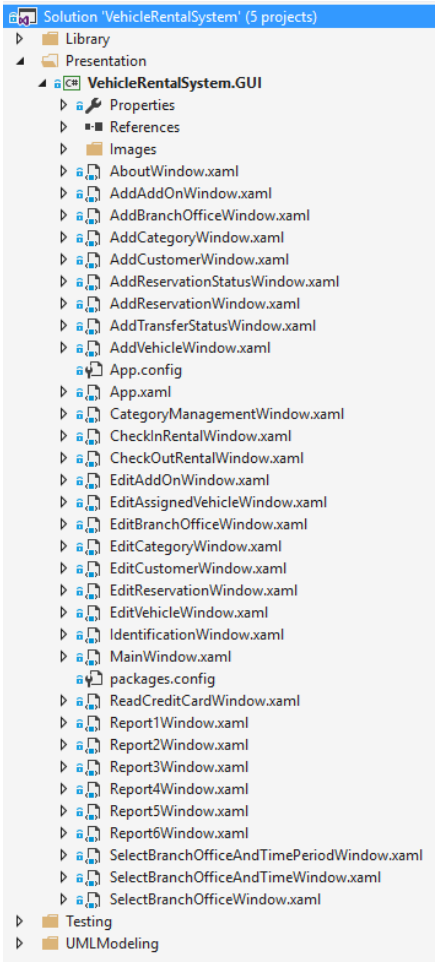

Ilustración 38: Capa de presentación

Tal y como se puede observar en la Ilustración 38, la capa de presentación está contenida en un proyecto independiente. Esta capa de presentación está formada por lo que en Visual Studio se conocen como páginas y por la carpeta contenedora de imágenes. El diseño de las páginas se puede definir tanto por medio de XAML como por medio de C#.

Por último, la carpeta de pruebas contiene en su interior dos proyectos distintos cuyo objetivo era el de comprobar el correcto comportamiento de las capas de persistencia y de lógica de negocio.

6.3 Código

Dado el gran volumen del número de funciones que contiene la aplicación, en este apartado se verán los aspectos más relevantes referentes a la implementación del código en cada una de las capas de la arquitectura de la aplicación.

6.3.1 Capa de presentación

La aplicación presenta un diseño típico ofrecido en los paquetes Office de Microsoft, donde las distintas funcionalidades ofrecidas vienen organizadas en distintas pestañas en la ventana principal. Por ello, con el fin de mejorar la organización y reducir el posible número de errores,

se ha utilizado la opción creación de regiones de código ofrecida por C#, donde cada región agrupa los métodos relacionadas con cada módulo.

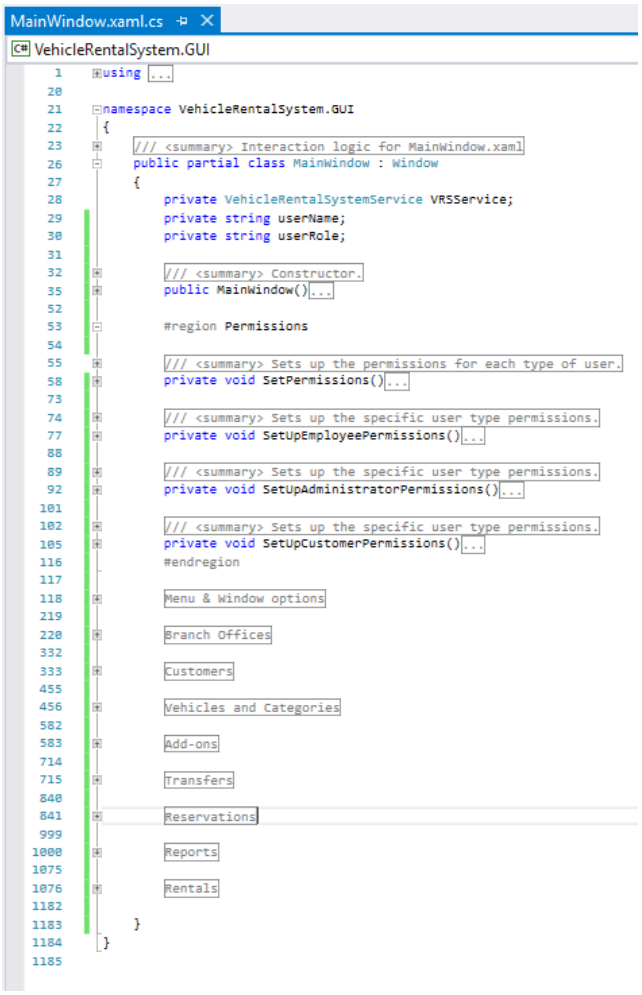

Ilustración 39: Código organizado en la ventana principal

Las operaciones más habituales en la aplicación son las de añadir, borrar, eliminar o buscar un elemento determinado en el módulo de gestión de oficinas, clientes, vehículos, extras, transferencias o alquileres. A excepción del método de búsqueda, el resto de los métodos implementados dentro de cada módulo poseen una estructura muy similar. El método de búsqueda puede diferir ligeramente entre los distintos módulos en función de si se va a buscar un elemento con un valor exacto o si se van a buscar los valores que empiecen por el valor de la cadena de texto introducida.

Considerando que la única función de la interfaz es obtener los datos introducidos por el cliente y mostrarle datos procesados, esta utiliza los métodos definidos a nivel de la capa de lógica. Un claro ejemplo que muestra dicho comportamiento es al añadir uno de los elementos:

```
        /// <summary> Constructor.
        public AddBranchOfficeWindow()...

        /// <summary>
        /// Checks that there are no missing values and creates a new branch office using the provided data.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void CreateButton_Click(object sender, RoutedEventArgs e)
        {
            if (CountryField.SelectedIndex != -1 && !String.IsNullOrEmpty(CityField.Text) &&
               !String.IsNullOrEmpty(PostalCodeField.Text) && !String.IsNullOrEmpty(AddressField.Text) &&
               !String.IsNullOrEmpty(PhoneNumberField.Text) && !String.IsNullOrEmpty(EmailField.Text))
            {
                try
                {
                    VRSService.AddBranchOffice(AddressField.Text, PostalCodeField.Text, PhoneNumberField.Text, (string)CountryField.SelectedItem, EmailField.Text, CityField.Text);
                    this.Close();
                }
                catch (ServiceException exception)
                {
                    MessageBox.Show(exception.Message, "Operation not performed.", MessageBoxButton.OK, MessageBoxImage.Error);

                }
            }
            else
            {
                MessageBox.Show("Please, fill in all the fields.", "Operation not performed.", MessageBoxButton.OK, MessageBoxImage.Error);
            }
        }
```

Ilustración 40: Método para añadir una nueva oficina

Otro claro ejemplo es el de cómo se implementan las operaciones de edición de un elemento determinado. En la Ilustración 41 se muestra el método de edición sobre un cliente que ha seleccionado el usuario.

```
/// <summary> Saves the modified customer data.
private void UpdateButton_Click(object sender, RoutedEventArgs e)
{
    if (!String.IsNullOrEmpty(FullNameField.Text) && !String.IsNullOrEmpty(IDField.Text) &&
       FirstRegistrationField.SelectedDate != null && !String.IsNullOrEmpty(AddressField.Text) &&
       !String.IsNullOrEmpty(CityField.Text) && !String.IsNullOrEmpty(PostalCodeField.Text) &&
       !String.IsNullOrEmpty(PhoneNumberField.Text) && !String.IsNullOrEmpty(CreditCardNumber.Text) &&
       !String.IsNullOrEmpty(CreditCardTypeField.SelectedValue as string) && !String.IsNullOrEmpty(CreditCardMonthField.Text) &&
       !String.IsNullOrEmpty(CreditCardYearField.Text) && !String.IsNullOrEmpty(CVCField.Text) &&
       !String.IsNullOrEmpty(UserNameField.Text))
    {
        try
        {
            VRSService.EditCustomer(this.SelectedCustomer.Id, FullNameField.Text, IDField.Text, (DateTime)FirstRegistrationField.SelectedDate, AddressField.Text, CityField.Text, PostalCodeField.Text, PhoneNumberField.Text,
        UserNameField.Text, (string)CreditCardTypeField.SelectedValue, CreditCardNumber.Text, Convert.ToInt32(CreditCardMonthField.Text), Convert.ToInt32(CreditCardYearField.Text), Convert.ToInt32(CVCField.Text));
            if ((bool)ResetPasswordCheckBox.IsChecked)
            {
                VRSService.ChangeUserAccountPassword(SelectedCustomer.UserAccount, PasswordTextBox.Password, RepeatPasswordTextBox.Password);
            }
            this.Close();
        }
        catch (ServiceException exception)
        {
            MessageBox.Show(exception.Message, "Operation not performed.", MessageBoxButton.OK, MessageBoxImage.Error);
        }
    }
    else
    {
        MessageBox.Show("Please, fill in all the fields. If you do not want to change the password, leave it empty.", "Operation not performed.", MessageBoxButton.OK, MessageBoxImage.Error);
    }
}
```

Ilustración 41: Método de edición de un cliente

Las funciones de borrado, edición y búsqueda de un elemento también presentan dicho comportamiento:

```
/// <summary> Handles the delete add-on button click.
private void DeleteAddOnButtonClick(object sender, RoutedEventArgs e)
{
    string confirmationString = "This operation will delete or retire the add-on depending on if it has been previously assigned to a reservation or not. Once it has been retired, it will not be available in the system for its use. Would you like to continue?";
    if (AddOnDataGrid.SelectedItem != null && AskForConfirmation("Delete add-on", confirmationString))
    {
        try
        {
            VRSService.DeleteRemovableExtra((RemovableExtra)AddOnDataGrid.SelectedItem);
            AddOnDataGrid.ItemsSource = LoadAddOnDataGrid();
        }
        catch (ServiceException exception)
        {
            MessageBox.Show(exception.Message, "Operation not performed.", MessageBoxButton.OK, MessageBoxImage.Exclamation);
        }
    }
}
```

Ilustración 42: Método de borrado de un extra

```
/// <summary> Checks that there are no missing values and updates the selected add-on using the provided data.
private void UpdateButton_Click(object sender, RoutedEventArgs e)
{
    if (CountryField.SelectedIndex != -1 && CityField.SelectedIndex != -1 &&
       AddressField.SelectedIndex != -1 &&
       !String.IsNullOrEmpty(NameField.Text) && !String.IsNullOrEmpty(PriceField.Text))
    {
        try
        {
            VRSService.EditRemovableAddOn(selectedAddOn.Id, NameField.Text, Convert.ToDouble(PriceField.Text), VRSService.FindBranchOfficeByAddress(AddressField.Text));
            this.Close();
        }
        catch (ServiceException exception)
        {
            MessageBox.Show(exception.Message, "Operation not performed.", MessageBoxButton.OK, MessageBoxImage.Error);

        }
    }
    else
    {
        MessageBox.Show("Please, fill in all the fields.", "Operation not performed.", MessageBoxButton.OK, MessageBoxImage.Error);
    }
}
```

Ilustración 43: Método de edición de un complemento

```
/// <summary> Handles the text changed event and works as a search bar.
private void VehicleFastSearchTextBox_TextChanged(object sender, TextChangedEventArgs e)
{
    TextBox fastSearchTextBox = (TextBox)sender;
    string vehiclePlateNumber = fastSearchTextBox.Text;
    ICollectionView collectionView = CollectionViewSource.GetDefaultView(VehicleDataGrid.ItemsSource);
    if (!string.IsNullOrEmpty(vehiclePlateNumber))
    {
        collectionView.Filter = o =>
        {
            Vehicle vehicle = o as Vehicle;
            return (vehicle.PlateNumber.StartsWith(vehiclePlateNumber));
        };
    }
    if (vehiclePlateNumber.Length == 0)
    {
        VehicleDataGrid.ItemsSource = LoadVehicleDataGrid();
    }
}
```

Ilustración 44: Método de búsqueda de un elemento

Tal y como se puede observar, ninguno de los métodos de las ilustraciones anteriores implementa lógica de negocio, sino que realizan llamadas a las funciones definidas en la capa de lógica de negocio tras realizar la captura de algún evento ocurrido en la interfaz de usuario. Esto mismo ocurre en todas las funciones definidas en la capa de presentación.

En cuanto al resto de métodos definidos en la capa de presentación, caben destacar los de validación de los valores introducidos en los campos, aquellos que cargan los atributos de un objeto en los campos de una pantalla de edición o los de los desplegables también conocidos como ComboBox o por último aquellos métodos que permiten la generación de los informes.

Un ejemplo de método de validación de los datos introducidos en una pantalla sería el que se presenta en la pantalla de inicio de sesión. El método mostrado en la Ilustración 45 comprueba si la contraseña tiene una longitud mínima de 8 caracteres, alguna mayúscula y algún número.

```
/// <summary> Checks if the introduced password is valid even before trying to log in. A valid password contains a cappi ...
private bool IsValidPassword(string password)
{
    if (password.Length != 0)
    {
        Regex hasNumber = new Regex(@"[0-9]+");
        Regex hasUpperChar = new Regex(@"[A-Z]+");
        Regex hasMinimum8Chars = new Regex(@".{8,}");

        return hasNumber.IsMatch(password) && hasUpperChar.IsMatch(password) && hasMinimum8Chars.IsMatch(password);
    }
    return false;

}
```

Ilustración 45: Método de validación de una contraseña introducida

Al seleccionar un elemento de los módulos mencionados anteriormente, existe la posibilidad de editar este mediante una pantalla. Los métodos de carga de atributos de un objeto en los campos de una pantalla presentan una estructura muy similar a la mostrada en la Ilustración 46, en el que se asignan los atributos del objeto seleccionado a los campos de la pantalla de edición.

```
/// <summary>
/// Displays the information of the selected vehicle.
/// </summary>
private void LoadAllScreenFields()
{
    PlateNumberField.Text = this.selectedVehicle.PlateNumber;
    MakeField.Text = this.selectedVehicle.Make;
    ModelField.Text = this.selectedVehicle.Model;
    ColourField.Text = this.selectedVehicle.Colour;
    FuelTypeField.Text = this.selectedVehicle.FuelType;
    FirstRegistrationField.SelectedDate = this.selectedVehicle.FirstRegistration;
    GearTypeField.Text = this.selectedVehicle.GearType;
    SeatsNumberField.Text = this.selectedVehicle.SeatsNumber.ToString();
    CategoryField.SelectedValue = this.selectedVehicle.Category.Name;
    PowerField.Text = this.selectedVehicle.Power.ToString();
    BranchOfficeField.Text = this.selectedVehicle.AssignedToBranchOffice.Address;
    CountryField.SelectedValue = this.selectedVehicle.AssignedToBranchOffice.Country;
    CityField.SelectedValue = this.selectedVehicle.AssignedToBranchOffice.City;
    BranchOfficeField.SelectedValue = this.selectedVehicle.AssignedToBranchOffice.Address;
}
```

Ilustración 46: Método de carga de los atributos de un objeto en la pantalla de edición

Por último, se encuentran todos aquellos métodos que posibilitan la generación, previsualización, impresión y descarga de un informe generado en la aplicación en formato PDF. Entre estos métodos, destaca el método de generación de la previsualización y el método de impresión o guardado del documento que se puede visualizar en la pantalla de previsualización.

El método de generación de la previsualización, primeramente, genera la tabla que contendrá los elementos del informe. A continuación, obtiene un listado de objetos mediante la utilización de uno de los métodos implementados en la capa de lógica para dicho fin. Tras ello, contabiliza el número de elementos a mostrar y considera si estos van a caber en una cara de un folio o si por el contrario múltiples caras deberán de ser generadas. Como último paso, genera la hoja o hojas necesarias de forma automática e introduce una cantidad de elementos determinada en cada una de ellas.

El método de guardado e impresión se ha implementado utilizando librerías estándar sin recurrir a librerías de terceros para lograr dicho fin. El funcionamiento de este consiste en que una vez el usuario selecciona el botón habilitado en la pantalla de previsualización, sale una ventana preguntando por la impresora a utilizar para imprimir o si por el contrario desea guardar el documento.

Para posibilitar las acciones de guardado en formato PDF o impresión, se consideran el tamaño de las hojas generadas en la pantalla de previsualización, se genera un documento virtual que contiene una o múltiples hojas y se le pasa a Windows para que ejecute la acción seleccionada por el empleado.

6.3.2 Capa de lógica de negocio

Esta capa presenta todas las clases vistas en el diagrama de clases durante la fase de análisis y define un modelo de datos con el que se trabaja por medio de servicios que vienen implementados por los métodos.

En la estructura de la aplicación mostrada en la Ilustración 37, la clase "VehicleRentalSystemService" implementa todos los servicios de la aplicación. Estos métodos vienen definidos en la interfaz "IVehicleRentalSystemService".

En esta capa, el código correspondiente a cada módulo está implementado y organizado dentro de la clase "VehicleRentalSystemService". La organización de los módulos se ha realizado por medio de las regiones que permite definir C#.

Ilustración 47: Implementación de los módulos organizada en regiones

Adicionalmente, esta clase implementa un patrón singleton sobre la capa de acceso a datos (DAL). Esto significa que la clase se ha implementado de manera que durante la ejecución del

programa únicamente puede existir una única instancia del objeto con la que interactuará para realizar las operaciones de lectura y escritura sobre la base de datos.

Dado el gran número de líneas de código, a continuación, se revisarán los patrones utilizados para definir los métodos más utilizados que permiten añadir, editar, eliminar o actualizar los objetos en los módulos de gestión de oficinas, clientes, vehículos, extras, transferencias y alquileres.

Para crear un nuevo objeto, dentro de cada módulo se ha implementado los métodos que siguen el mismo patrón que el utilizado en el módulo de vehículos mostrado en la Ilustración 48.

```
#region CARS - Cars Management Module
/// <summary> Adds a new vehicle to the database.
public void AddVehicle(Vehicle vehicle)
{
    dal.Insert<Vehicle>(vehicle);
    dal.Commit();
}

/// <summary> Adds a new vehicle to the database.
public void AddVehicle(string plateNumber, string make, string model, string colour, string fuelType, DateTime firstRegistration,
    string gearType, int seatsNumber, int power, Category category, BranchOffice branchOffice)
{
    if (!ExistsVehicle(plateNumber))
    {
        if (IsVehicleDataValid(firstRegistration, power, seatsNumber))
        {
            Vehicle vehicle = new Vehicle(plateNumber, make, model, colour, fuelType, firstRegistration,
            gearType, seatsNumber, power, category, branchOffice);
            AddVehicle(vehicle);
        }
    }
    else throw new ServiceException("This vehicle already exists.");
}
```

Ilustración 48: Método para añadir un nuevo vehículo

El patrón consiste en comprobar la validez de los datos proporcionados para crear el objeto y en caso de que estos sean válidos se crea la instancia y se llama a un método para que lo inserte y guarde en la base de datos.

Los métodos de edición de objetos ya existentes en la base de datos también siguen el patrón detallado a continuación.

```
/// <summary> Modifies a customer in the database.
public void EditCustomer(int id, string fullName, string personalID, DateTime drivingLicenseDate, string address, string city, string postalCode, string phoneNumber,
        string userName, string creditCardType, string creditCardNumber, int creditCardMonth, int creditCardYear, int cvc)
{
    if (IsCustomerNewDataValid(drivingLicenseDate, creditCardType, creditCardNumber))
    {
        Customer customer = FindCustomerByID(id);
        customer.FullName = fullName;
        customer.PersonalID = personalID;
        customer.DrivingLicenseDate = drivingLicenseDate;
        customer.Address = address;
        customer.City = city;
        customer.PostalCode = postalCode;
        customer.PhoneNumber = phoneNumber;
        customer.UserAccount.UserName = userName;
        customer.CreditCard.Type = creditCardType;
        customer.CreditCard.Digits = creditCardNumber;
        customer.CreditCard.Month = creditCardMonth;
        customer.CreditCard.Year = creditCardYear;
        customer.CreditCard.Cvc = cvc;
        dal.Commit();
    }
}
```

Ilustración 49: Método para la edición de un cliente

El patrón que siguen los métodos de edición en esta capa será el mostrado en la Ilustración 49 que consiste en la localización del objeto dentro de la base de datos, en la reasignación de todos

los atributos de este a excepción del identificador y en realizar el guardado del objeto modificado en la base de datos.

El borrado de un objeto existente en la base de datos viene implementado también siguiendo un patrón bien definido. Un ejemplo de implementación de un método de borrado es el que aparece en la Ilustración 50.

```
/// <summary> Deletes a removable extra from the database if the extra has not been assigned to any reservation. Otherwi ...
public void DeleteRemovableExtra(RemovableExtra removableExtra)
{
    if (removableExtra != null)
    {
        try
        {
            if (ExistsRemovableAddOn(removableExtra))
            {
                if (ExistsReservationWithRemovableExtra(removableExtra))
                {
                    removableExtra.IsRetired = true;
                }
                else
                {
                    dal.Delete<RemovableExtra>(removableExtra);
                }
                dal.Commit();
            }
        }
        catch (ArgumentNullException)
        {
            throw new ServiceException("This add-on does not exist.");
        }
    }
}
```

Ilustración 50: Método de borrado de un extra

Las pautas que siguen los métodos de borrado consisten en comprobar si el objeto indicado existe en la base de datos. En dicho caso, en función del objeto del que se trate, se borra directamente de la base de datos o se comprueba si el objeto ya se ha utilizado o asignado alguna vez, lo que significará que contiene información importante para una reserva. En este caso, el objeto se marca como retirado en la base de datos de manera que dejaría de estar disponible para su posterior uso, pero no se perdería su información.

En cuanto al método de búsqueda de un elemento, este se ha definido de diversas formas en cada módulo existente, dado que los atributos por los que se realiza la búsqueda van cambiando de un módulo a otro.

Por último, la capa de lógica de negocio también incluye una definición parcial de las clases. Estas contienen los constructores por defecto y generales de cada clase.

6.3.3 Capa de persistencia

Esta capa tiene la función de leer y guardar toda la información con la que trabaja la aplicación. En la aplicación, será la encargada de manejar toda aquella información con la que trabajen cada uno de los módulos desarrollados que componen esta.

La Ilustración 37 muestra los ficheros que forman parte de esta capa.

Por una parte, se encuentra la implementación parcial de las clases definidas en el diagrama de clases. Esta consiste en la implementación de los atributos indicados en el diagrama de clases. Se ha utilizado dicha técnica con el fin de evitar problemas de sobreescritura de clases ya existentes tras realizar cualquier cambio en el modelo y volver a generar las clases.

Por otra parte, en esta capa se implementarán los patrones “Repository” y “Unit Of Work” que definen el acceso a los datos. El patrón se puede observar en la Ilustración 51.

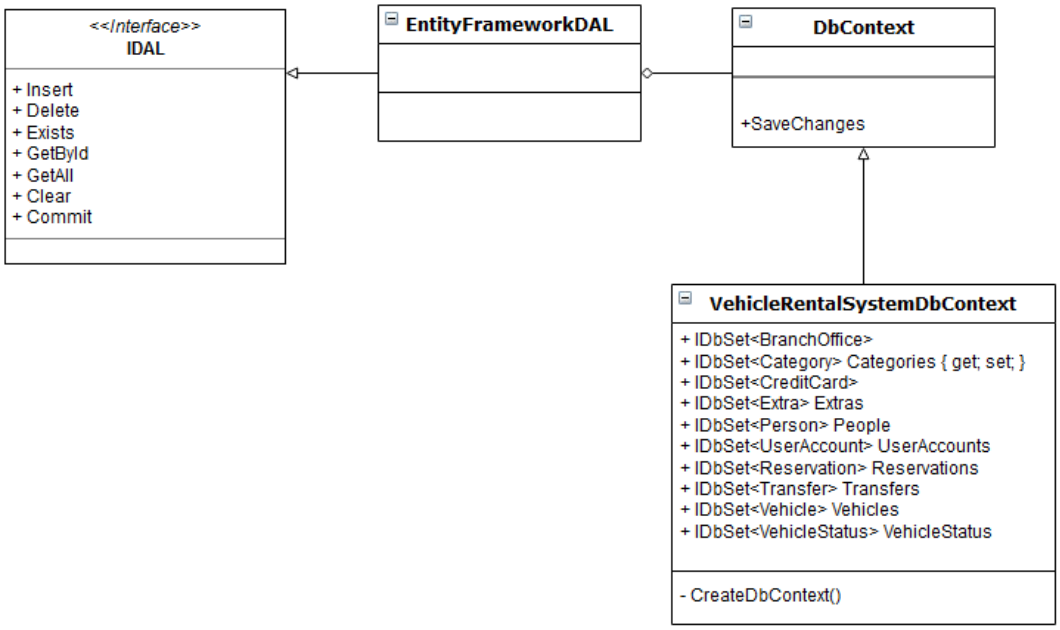

Ilustración 51: Arquitectura de la capa de acceso a datos

Existe por tanto una interfaz llamada “IDAL” que define todas aquellas operaciones en forma de métodos y son accesibles desde la capa de lógica de negocio.

La clase “EntityFrameworkDAL” es la encargada de implementar las operaciones en forma de métodos genéricos que forman parte de la implementación del patrón DAO.

Por otra parte, la clase llamada “VehicleRentalSystemDbContext” extiende la clase llamada DbContext creando así lo que se conoce como la unidad de trabajo y define los repositorios de datos de tipo T que permitirán consultar, añadir o eliminar objetos en la base de datos

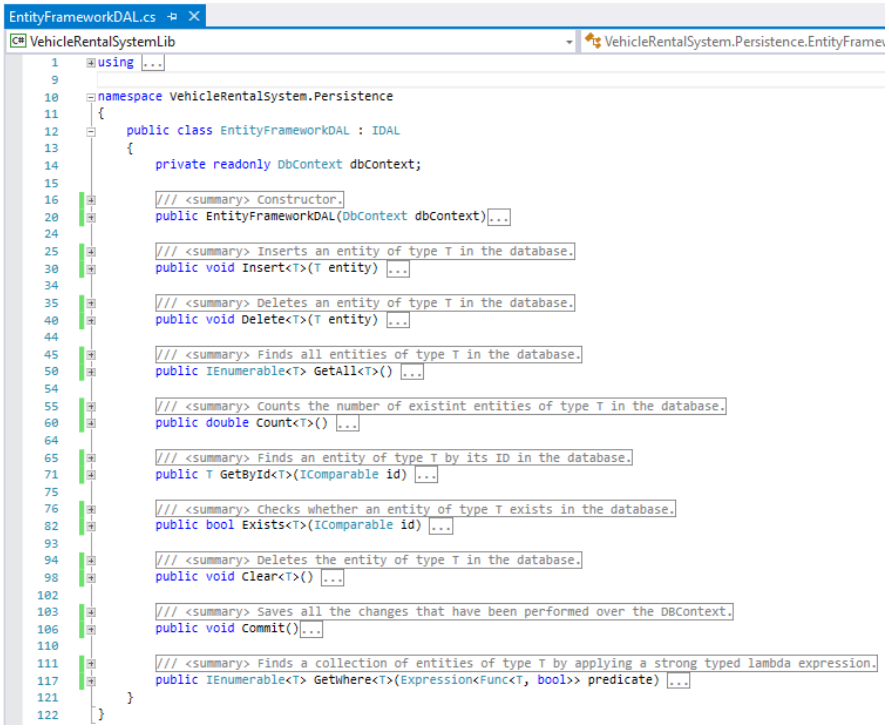

Ilustración 52: Clase "EntityFrameworkDAL"

7. Pruebas

Las pruebas realizadas para verificar que la solución funciona correctamente se han implementado en dos proyectos distintos contenidos dentro de la misma solución. Estos se muestran en la Ilustración 53.

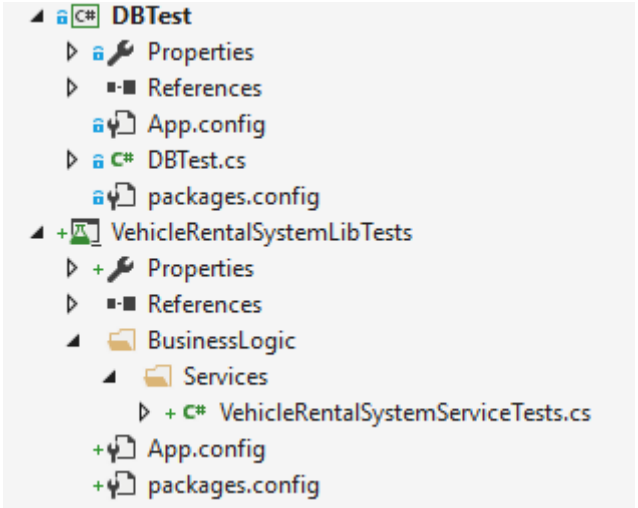

Ilustración 53: Proyectos para la fase de pruebas

7.1 Pruebas sobre la capa de persistencia

El proyecto "DBTest" implementa una serie de métodos con el fin de probar que la funcionalidad ofrecida por la capa de persistencia y reflejada en la interfaz IDAL funciona correctamente. En la Ilustración 54 refleja la agrupación del código en regiones, organizando dichos métodos para lograr un código limpio.

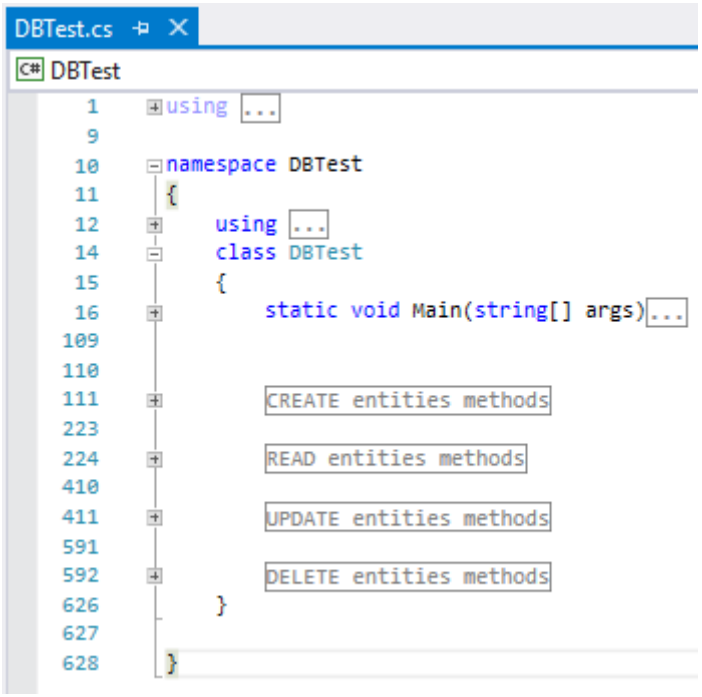

Ilustración 54: Regiones en las que se clasifican los métodos de prueba

El correcto funcionamiento se comprueba creando primeramente todas las entidades posibles y efectuando su guardado en la base de datos. Posteriormente, todas las entidades son recuperadas de la base de datos, modificadas y vueltas a guardar en la base de datos. Por último, se aplica el proceso de borrado sobre todas estas.

Durante la ejecución del programa, tal y como se puede observar en la Ilustración 55, se puede observar el flujo de ejecución del programa durante el que se muestran las acciones realizadas en cada momento. Tras la ejecución, se pueden contrastar los cambios efectuados sobre la base de datos mediante la apertura y visualización del contenido de la base de datos en Visual Studio.

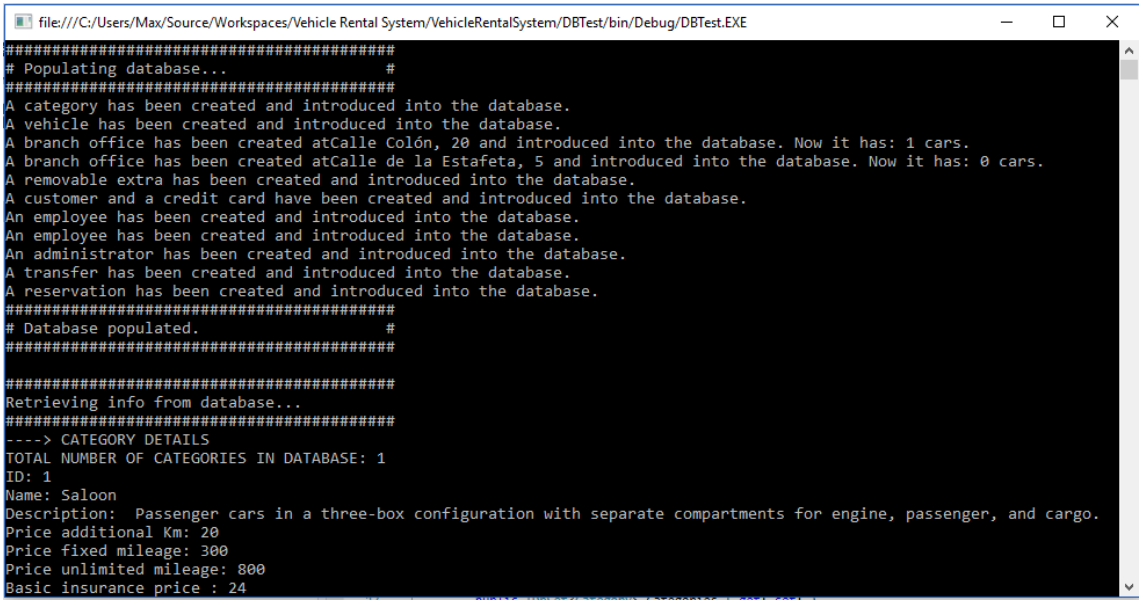

Ilustración 55:Flujo de ejecución de la prueba

7.2 Pruebas sobre la capa de lógica de negocio

Para realizar las pruebas sobre la capa de lógica de negocio se ha implementado un nuevo proyecto llamado “VehicleRentalSystemLibTests”. Este proyecto implementa una serie de pruebas unitarias sobre todos los métodos ofrecidos por la interfaz “IVehicleRentalSystemService”.

Visual Studio proporciona un explorador de pruebas unitarias que permite visualizar, seleccionar y ejecutar las pruebas unitarias creadas. Los conceptos básicos se pueden encontrar en la página web oficial (6).

Un claro ejemplo de prueba unitaria realizada viene siendo el mostrado en la Ilustración 56. La función de esta es comprobar que el método añade correctamente una categoría a la base de datos. Una vez realizada la operación, se comprueba que la acción se ha realizado correctamente y la prueba aparece marcada como válida en el explorador de pruebas unitarias de la forma mostrada en la Ilustración 57.

```
[TestMethod()]
public void AddCategoryTest()
{
    IVehicleRentalSystemService VRSService = VehicleRentalSystemService.getVehicleRentalSystemService();
    VRSService.RemoveAllData();

    VRSService.AddCategory("Luxury", 2, 12, 23, "Luxury vehicles.", 80, 120, null, new List<Vehicle>());

    Category luxuryCategory = VRSService.FindCategoryByName("Luxury");
    Assert.AreEqual(luxuryCategory.Name, "Luxury");
    Assert.AreEqual(luxuryCategory.PriceAdditionalKm, 2);
    Assert.AreEqual(luxuryCategory.PriceFixedMileage, 12);
    Assert.AreEqual(luxuryCategory.PriceUnlimitedMileage, 23);
    Assert.AreEqual(luxuryCategory.Description, "Luxury vehicles.");
    Assert.AreEqual(luxuryCategory.BasicInsurancePrice, 80);
    Assert.AreEqual(luxuryCategory.ComprehensiveInsurancePrice, 120);
    Assert.IsNull(luxuryCategory.UpperCategory);
    Assert.ReferenceEquals(luxuryCategory.Vehicles, new List<Vehicle>());

    VRSService.AddCategory("Sedan", 2, 10, 22, "Comfortable sedan vehicles.", 40, 60, VRSService.FindAllCategories().First(), new List<Vehicle>());

    Category sedanCategory = VRSService.FindCategoryByName("Sedan");
    Assert.AreEqual(sedanCategory.Name, "Sedan");
    Assert.AreEqual(sedanCategory.PriceAdditionalKm, 2);
    Assert.AreEqual(sedanCategory.PriceFixedMileage, 10);
    Assert.AreEqual(sedanCategory.PriceUnlimitedMileage, 22);
    Assert.AreEqual(sedanCategory.Description, "Comfortable sedan vehicles.");
    Assert.AreEqual(sedanCategory.BasicInsurancePrice, 40);
    Assert.AreEqual(sedanCategory.ComprehensiveInsurancePrice, 60);
    Assert.ReferenceEquals(sedanCategory.UpperCategory, VRSService.FindAllCategories().First());
    Assert.ReferenceEquals(sedanCategory.Vehicles, new List<Vehicle>());

    VRSService.RemoveAllData();
}
```

Ilustración 56: Ejemplo prueba unitaria sobre el método de añadir categoría

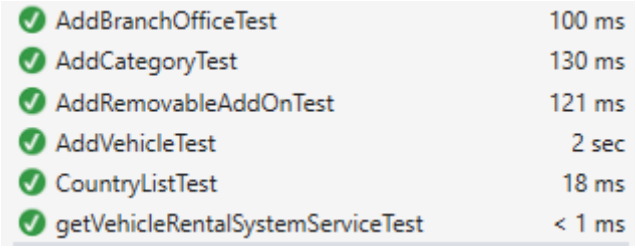

Ilustración 57: Explorador de pruebas unitarias de Visual Studio

7.3 Pruebas sobre la capa de presentación

Las pruebas realizadas en este apartado tienen como objetivo verificar que la interfaz gráfica diseñada se visualiza correctamente y en que todas las acciones que se puedan realizar están asignadas correctamente a eventos que al ejecutarse darán lugar a llamadas sobre los servicios ofrecidos en forma de métodos en la interfaz “IVehicleRentalSystemService” por la capa de lógica de negocio.

Estas pruebas se han realizado manualmente tras la creación de cada una de las ventanas realizando todas aquellas acciones que vienen definidas en los casos de uso y verificando que todo funciona correctamente.

8. Conclusiones

Este último apartado pone en manifiesto que los objetivos definidos inicialmente en la aplicación se han alcanzado de manera satisfactoria.

Esto se ha logrado tras haber realizado una planificación detallada de las tareas a realizar a lo largo de todo el proyecto en base a los objetivos iniciales, de las tecnologías a utilizar y de las estrategias a seguir durante la fase de diseño. Esto ha evitado problemas de organización y tiempo en el ámbito de la gestión del proyecto.

Para la creación de la aplicación se ha utilizado lo que se conoce como un ciclo de vida tradicional en la ingeniería del software. Esto ha sido posible dado que los objetivos a alcanzar estaban bien definidos desde un principio y a diferencia con lo que suele ocurrir en el mundo laboral, estos no han variado con el paso del tiempo durante la realización del proyecto. En este último caso, se hubiera optado por una metodología ágil para hacer frente a dicho problema.

8.1 Relación del trabajo desarrollado con los estudios cursados

Con el desarrollo del presente proyecto, se ha mejorado la capacidad organizativa en el ámbito de la gestión de proyectos ya adquirida durante la carrera en la asignatura de “Gestión de proyectos”.

Adicionalmente se han mejorado los conocimientos existentes adquiridos en la asignatura de “Ingeniería del software” sobre las tecnologías utilizadas de manera significativa, dado que en este último caso se ha realizado un proceso de búsqueda de información para completar y mejorar los conocimientos parciales previamente existentes sobre estas. Cabe destacar que en el proyecto se ha utilizado un proceso de creación de la aplicación (proceso software) visto también en detalle en la asignatura de rama de Ingeniería del software llamada “Diseño software”. Además, se han puesto en práctica los conocimientos adquiridos en esta última para crear un código limpio.

Las competencias transversales que han sido requeridas para la realización del presente trabajo han sido la habilidad de toma de decisiones, la capacidad de gestión y planificación, la capacidad de resolución de problemas y la creatividad para la etapa de diseño. Todas estas han sido necesarias en un alto grado para la correcta planificación y desarrollo del proyecto.

9. Trabajos futuros

En un futuro se podría mejorar la aplicación añadiendo más módulos de gestión según aparezca la necesidad dentro de la empresa.

Una posible mejora de la aplicación sería la implementación de un módulo de gestión de empleados que tuviese información detallada sobre estos y sobre los horarios de trabajo que estos realizan. Adicionalmente, este podría mostrar estadísticas sobre el número de vehículos transferidos y sobre el historial del empleado con respecto a los vehículos transferidos o posibles desperfectos causados accidentalmente por este.

Otra posible mejora sería la implementación de extras no removibles. Actualmente los vehículos incluyen una serie de extras integrados que no se pueden extraer como el GPS, asistente en cuesta y en frenada, reconocimiento de señales o el cambio automático de manera que cuando se formalizase una reserva se pudiesen incluir dichas preferencias para buscar un vehículo en la categoría seleccionada que cumpliese con los criterios seleccionados y este se asignase con un coste extra adicional por cada extra integrado seleccionado. En caso de no existir ninguna preferencia por parte del cliente y de que el vehículo fuese finalmente asignado por cualquier motivo a la reserva, esto no influiría en el precio del alquiler indicado en la categoría.

Por último, otro posible trabajo futuro sería realizar la aplicación de manera que esta sea distribuida y multidispositivo. Para lograr que sea distribuida, en primer lugar, se utilizaría una arquitectura cliente/servidor. De esta forma, habría que disponer de una base de datos remota accesible por medio de internet sobre la cual se pudiesen lanzar solicitudes asíncronas a través de un socket configurado en nuestra aplicación. Por ello, habría que realizar cambios sobre la capa de persistencia. Adicionalmente, para conseguir que sea multidispositivo se podría crear una aplicación web que utilizase la base de datos remota. De esta manera, el usuario final podría acceder desde cualquier dispositivo.

10. Referencias

1. **Factultad de informática de la UCM.** Especificación de requisitos según el estándar IEEE 830-1998. [En línea] HTTPS://WWW.FDI.UCM.ES/PROFESOR/GMENDEZ/DOCS/IS0809/IEEE830.PDF.

2. **Sommerville, Ian.** *Ingenieria de Software.* Novena edición. s.l. : PEARSON, 2011. 978-607-32-0603-7.

3. **Organización Internacional de Normalización (ISO).** ISO/IEC 25010. *Calidad del producto software.* [En línea] https://iso25000.com/index.php/en/iso-25000-standards/iso-25010.

4. **Inheritance with EF Code First: Part 1 – Table per Hierarchy (TPH). [En línea] HTTPS://WEBLOGS.ASP.NET/MANAVI/INHERITANCE-MAPPING-STRATEGIES-WITH-ENTITY-FRAMEWORK-CODE-FIRST-CTP5-PART-1-TABLE-PER-HIERARCHY-TPH .**

5. Microsoft docs. Windows Presentation Foundation. [En línea] HTTPS://DOCS.MICROSOFT.COM/EN-US/DOTNET/FRAMEWORK/WPF/.

6. —. Conceptos básicos de las pruebas unitarias. [En línea] https://docs.microsoft.com/es-es/visualstudio/test/unit-test-basics?view=vs-2019Conceptos básicos de las pruebas unitarias.