Aplicación móvil para la Gestión del Ciclo de Venta de Automóviles de un concesionario

Proyecto creado por Leandro Cieri

Supervisado por:
Pablo Carmelo Fernández López

Fecha:
20 de Junio de 2022

Tabla de contenido

Agradecimientos

Este trabajo no podría haberse realizado sin la inmensurable ayuda de mi familia, amigos, compañeros y profesores que no solo me han apoyado durante el transcurso del proyecto, si no de toda la carrera, haciéndome ver que el esfuerzo merecería la pena y esforzándose ellos cuando uno ya no podía.

Sin ellos este trabajo no habría sido posible, así que, a todos esos coautores y coautoras honorarios, gracias.

Resumen

Actualmente, en el mercado de compraventa de vehículos de ocasión existe una alta competencia, mayoritariamente entre pequeñas y medianas empresas que ofrecen productos y servicios muy similares. Sobre esta premisa nace este proyecto, para crear una herramienta con la que una compañía pueda diferenciarse de cara al cliente y realizar sus procesos de negocio de una forma mucho más eficaz.

La herramienta consta de una aplicación móvil destinada al uso de los clientes, actuales o potenciales, del concesionario, con la que podrán acceder a toda la información de los vehículos adquiridos u ofertados. Por otra parte, los empleados del concesionario dispondrán de un programa con el que registrar y consultar todas las transacciones que realizan en sus trabajos.

A la solución se ha llegado mediante la obtención y especificación de requisitos, creación de conceptos y entrega de múltiples iteraciones en las que el producto iba sumando cualidades y funciones, adaptándose a nuevas necesidades y fluctuaciones de prioridades, siguiendo en todo momento la metodología de desarrollo ágil.

Abstract

Currently, there is a high level of competition in the second-hand car market, mainly between small and medium-sized companies that offer very similar products and services. On this premise, this project is made to create a tool with which a company can differentiate itself and carry out its business processes in a much more efficient way.

The tool consists of a mobile application for use by current or potential customers of the dealership, with which they can access all the information on the vehicles purchased or offered. On the other hand, the dealership's employees will have a programme with which to register and consult all the transactions they carry out in their jobs.

The solution has been reached by obtaining and specifying requirements, creating concepts and delivering multiple iterations in which the product acquired qualities and functions, adapting to new needs and fluctuations in priorities, following the agile development methodology at all times.

1 Introducción

1.1 Motivación

En España, en 2021 se vendieron 1.989.662 unidades de vehículos de ocasión, un 131% más que la cantidad de vehículos nuevos que se han matriculado en el mismo periodo (859.477). Por tanto, por cada vehículo nuevo que alguien compra en España, se venden 2,3 vehículos de segunda mano. (Ganvan, 2021)

Estos datos evidencian la magnitud del mercado de vehículos de ocasión, donde miles de concesionarios compiten entre ellos ofreciendo a clientes un bien cuya fabricación les es imposible de alterar, ya que depende de las propias marcas e importadores. Esto genera que acaben ofreciendo un producto y servicio muy similar entre ellos.

La similitud entre concesionarios hace que se puedan crear soluciones prácticamente universales, que casi cualquier empresa dedicada a la compraventa de vehículos de ocasión podría utilizar para agilizar tareas cotidianas. De ahí nace este proyecto, constituido con el fin de que estas empresas dispongan de una herramienta con la cuál sus procesos de negocio se realicen más rápido, con menos esfuerzo y ofreciendo una mejor calidad percibida para el cliente final.

La intención desde el inicio fue aprovechar esas semejanzas para estructurar un proyecto base, sobre el que se pueda crear un producto más personalizado y adaptado a las necesidades de cada empresa. Se pretende ofrecer un término medio entre producto y servicio, ya que un cliente adquiriría una serie de aplicaciones con funcionalidades predeterminadas y además la posibilidad de adaptarlas o extenderlas a su negocio concreto.

Por suerte, para la realización de este proyecto se ha contado con la ayuda de la empresa RCARS, un concesionario de vehículos de ocasión situado en la isla de Fuerteventura. Esta empresa, cuyos representantes han actuado como Product Owners durante el desarrollo, ha proporcionado una idea sobre las necesidades que querían subsanar y los procesos que querían agilizar, además, se ha realizado un trabajo de consultoría, identificando nuevas actividades que los stakeholders podrían realizar de una forma más eficiente con la ayuda de un software específico.

Su idea a priori era ofrecer a los clientes una app móvil, sin si quiera tener muy claro el contenido o la funcionalidad de la misma. Querían diferenciarse de la competencia ya que parte de su imagen de marca depende de ofrecer un servicio de una alta calidad dado que ofrecen vehículos de alta gama. Tras varias reuniones fueron quedando más claras las ideas y se acordó que la mejor solución sería digitalizar toda la información con la que trabajan, almacenarla en una base de datos y crear un sistema de acceso a esta, la base con funcionalidades predeterminadas anteriormente mencionada.

Esto ayudaría mucho a sus trabajadores ya que para realizar su trabajo dependían de archivadores físicos o, en el mejor de los casos, de hojas Excel improvisadas que mutaban con el tiempo en cientos de líneas sin mucho formato. Si otro concesionario o un cliente necesitaba una copia de un documento, se le mandaba una foto sacada con el móvil en ese momento. Si alguien necesitaba saber que coches había en stock, tenía que pedir que le enviaran un Excel que un trabajador creó *motu proprio* llamado "Listado de coches - copia -copia (3) (2)" y que mantenía actualizado en los momentos que se encontraba ocioso.

Por otra parte, como requisito personalizado, querían ofrecer algo que pudieran utilizar los clientes directamente. Estos descargarían una aplicación en sus teléfonos móviles donde podrían ver información de la tienda, de los coches que hay en venta y de los vehículos que han adquirido si era el caso. El objetivo principal de la app era reducir el número de llamadas telefónicas que tenían que atender los vendedores y que los clientes se relacionaran más con la marca.

1.2 Planificación

Para la realización de lo anteriormente mencionado, se han respetado, con cierto margen de error, las horas estimadas que se pueden ver en la siguiente tabla.

Fases	***Duración Estimada (horas)***	***Tareas (nombre y descripción, obligatorio al menos una por fase)***
Declaración Explicita del Problema	25 horas	Tarea 1.1: Análisis e Identificación de Requisitos Funcionales que debe cumplir el Sistema Software.
		Tarea 1.2: Estudio del Estado de la Cuestión e identificación de Sistemas de funcionalidades similares, para realizar la Comparativa.
Elaboración del Modelo del Sistema	60 horas	Tarea 2.1: Modelo de Análisis del Sistema. Construcción de los diferentes Metamodelos (Diagrama de Clases, Interacción entre Componentes, etc.) del Software del Sistema.
		Tarea 2.2: Modelo de Diseño y Arquitectura del Sistema. Construcción de los diferentes Metamodelos (Diagrama de Clases, Interacción entre Componentes, etc.) del Software del Sistema
Construcción y Realización de Pruebas del Sistema	190 horas	Tarea 3.1: Modelo de Pruebas, Modelo de Implementación y Modelo de Despliegue Construcción del Sistema Software.
Estudio de Resultados, Comparativa con Sistema similar y Documentación	25 horas	Tarea 4.1: Análisis y estudio de las métricas establecidas para el funcionamiento del Sistema.
		Tarea 4.2: Comparativa cualitativa de nuestro Sistema Software con Sistemas similares.
		Tarea 4.3.: Documentación y formalización del Proyecto (Memoria del TFG)

Ilustración 1 - Planificación inicial

La mayor diferencia entre los preestablecido y la realidad fue que no se tuvieron en cuenta las reuniones con los representantes, trabajadores y clientes de RCARS, que cabrían dentro de la Tarea 1.1 y 1.2 y por tanto estas llevaron considerablemente más tiempo.

Aun así, la totalidad del trabajo se ha realizado cumpliendo los requisitos impuestos por la ULPGC por los que se debe dedicar 300 horas en total.

1.3 Competencias y objetivos

El objetivo académico de este trabajo es la involucración en todos los aspectos y etapas del ciclo de vida de un software hecho a medida. Desde la obtención de requisitos hasta el mantenimiento pasando por el desarrollo y las pruebas, todas las fases del desarrollo se han trabajado en mayor o menor medida en este proyecto y por tanto se ha evidenciado

lo aprendido durante el grado de Ingeniería informática, más concretamente en la rama de Ingeniería del software.

De una forma más específica, se nombran a continuación las competencias concretas estipuladas por la ULPGC cuya relación con este trabajo es mayor:

IS01	Capacidad para desarrollar, mantener y evaluar servicios y sistemas software que satisfagan todos los requisitos del usuario y se comporten de forma fiable y eficiente, sean asequibles de desarrollar y mantener y cumplan normas de calidad, aplicando las teorías, principios, métodos y prácticas de la ingeniería del software.

Durante todo el desarrollo, la principal fuerza en la toma de decisiones fue la necesidad de cumplir con los requisitos de usuario y satisfacer a los clientes. Siempre usando métodos y prácticas correctas y que generen un producto de calidad.

IS02	Capacidad para valorar las necesidades del cliente y especificar los requisitos software para satisfacer estas necesidades, reconciliando objetivos en conflicto mediante la búsqueda de compromisos aceptables dentro de las limitaciones derivadas del coste, del tiempo, de la existencia de sistemas ya desarrollados y de las propias organizaciones.

Como se ha mencionado anteriormente, la obtención de requisitos, así como el refinamiento de los mismos supuso más tiempo de lo que se estimaba a priori, aunque el fruto de esto fue un compromiso en el que ambas partes estaban satisfechas con los costes, funcionalidades, tiempos y desarrollos futuros acordados.

IS03	Capacidad de dar solución a problemas de integración en función de las estrategias, estándares y tecnologías disponibles

Una baza importante de esta solución es la modularidad de las diferentes aplicaciones, combinando diversas tecnologías con el fin de poder incrementar las capacidades de una parte concreta del software sin afectar a las demás o añadir otros complementos que utilicen las herramientas ya implementadas. Se facilita mucho la expansión tanto horizontal como vertical del ámbito de este software.

IS04	Capacidad de identificar y analizar problemas y diseñar, desarrollar, implementar, verificar y documentar soluciones software sobre la base de un conocimiento adecuado de las teorías, modelos y técnicas actuales.

Prácticamente durante todo el transcurso del trabajo, se han realizado tareas coincidentes con los conocimientos descritos en esta competencia, especialmente en la parte donde se ha ejercido como consultor de soluciones TI para la empresa RCARS.

IS05	Capacidad de identificar, evaluar y gestionar los riesgos potenciales asociados que pudieran presentarse.

En la sección "Identificación de riesgos y mitigaciones" se puede ver el trabajo que se ha realizado relacionado con esta competencia.

2 Análisis

2.1 Estado del arte

A día de hoy, existen en el mercado aplicaciones con fines similares, gestionar un sistema de concesionarios además de las relaciones del mismo con sus agentes externos como pueden ser distribuidores de recambios, talleres o importadores de vehículos. Muchos sistemas aprovechan la alta integración en los procesos de negocio y el manejo de los datos de la empresa para además ofrecer herramientas de gestión financiera y de contabilidad.

Normalmente se centran en el apartado logístico, en la generación de informes y en la supervisión del estado del stock. Sin embargo, estas aplicaciones suelen optar a un nicho de mercado compuesto por empresas de un tamaño considerable, normalmente de vehículos nuevos, con diferentes concesionarios físicos, que tratan con una gran red de talleres y con unos niveles de ventas que necesiten de su propio departamento de contabilidad.

En el caso de un concesionario con un volumen de ventas menor, para obtener una herramienta que no exceda sus ámbitos empresariales y que se ajuste lo suficiente a sus procesos de negocio, se ha de recurrir a software hecho a medida, de hecho, en el caso concreto de RCARS, habían auditado posibles soluciones previas, pero todas se centraban

en herramientas de gestión cuyos usuarios serían los propios vendedores de la tienda, mientras que ellos buscaban que sea algo mucho más visible para el cliente final.

2.2 Estado de la cuestión y del mercado

Este producto, al ser altamente personalizado para la integración en el funcionamiento de una empresa concreta, disfruta de muy poca competencia, no se encuentran en el mercado herramientas muy similares que se puedan adaptar para cumplir los requisitos de los stakeholders. De hecho, la empresa quiere ofrecer este servicio para diferenciarse de la competencia, ya que prácticamente ningún otro concesionario de vehículos en canarias posee un servicio similar.

Lo más cercano en este ámbito, es la aplicación llamada “MyDAG” que ha desarrollado la empresa AIDA, perteneciente al Grupo Domingo Alonso.

Las semejanzas de esta aplicación con la que se quiere desarrollar en este trabajo son varias, permite conocer información superficial sobre vehículos que hayamos adquirido en uno de los concesionarios del grupo Domingo Alonso y además ofrece la posibilidad de pedir cita previa en uno de sus talleres.

Por otra parte, también consta de una sección de noticias en la que se muestran post relacionados con las empresas que componen el grupo, así como también ofrece unirse al “club WAH!” que trata de un sistema de fidelización en el que se obtienen descuentos en empresas asociadas como alquiler de vehículos o servicios de lavado. Sin embargo, no ofrece la posibilidad de pedir cita en un concesionario, de contactar con los empleados, de acceder a la documentación del vehículo, ni de ver los coches disponibles.

Como puede verse, las diferencias entre nuestro producto y MyDAG son más abundantes que las similitudes, y más aún si tenemos en cuenta que esta aplicación solo está disponible para los concesionarios del grupo que venden vehículos nuevos. Esto, sumado al hecho de que es una aplicación desarrollada de forma interna en una de sus empresas subsidiarias y que no se ofrece al mercado externo, deja a nuestro producto en una posición ventajosa en cuanto a la situación de mercado.

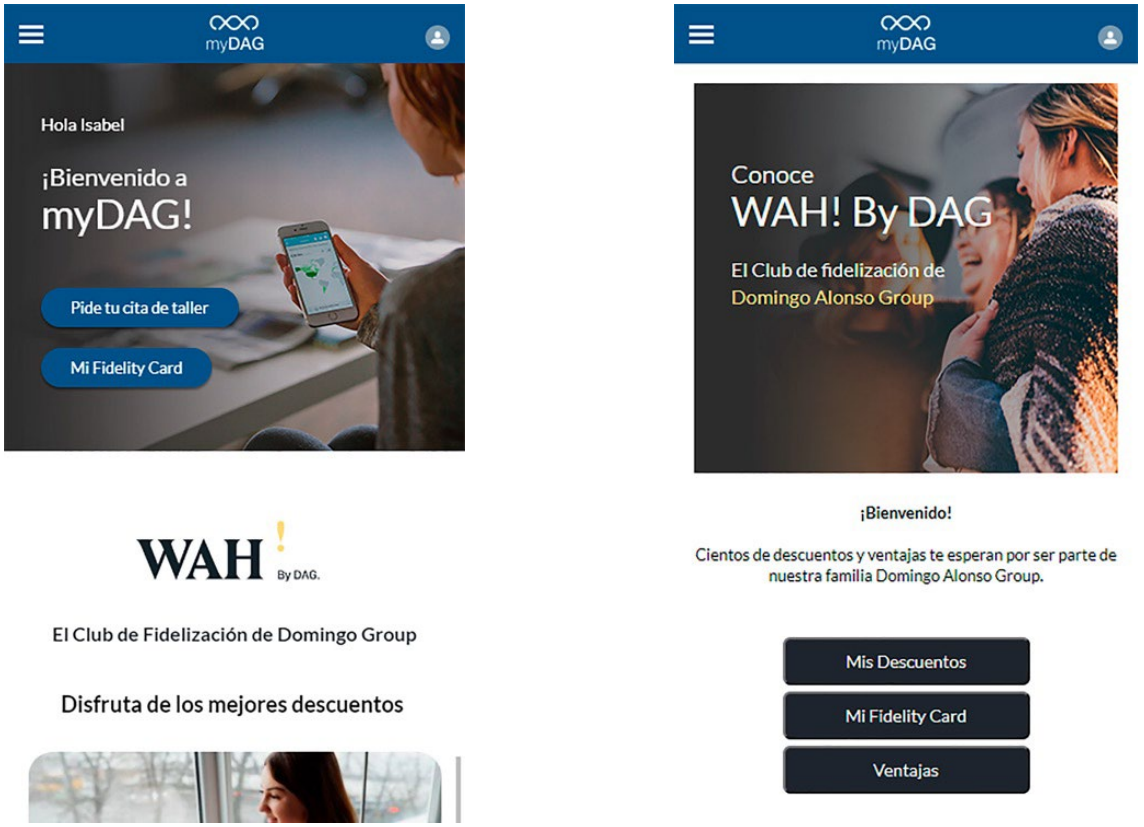


Ilustración 2 - Capturas de myDAG!

Otra herramienta similar, que es más común en este sector y que ya existe un número considerable de sistemas similares, es DealerCenter, como ellos mismos indican, "es un sistema de gestión de distribuidores centralizado y basado en la nube, diseñado para respaldar las operaciones diarias de concesionarios de automóviles independientes, incluidos de automóviles nuevos y usados. Con integración móvil con aplicaciones móviles nativas de DealerCenter para plataformas iOS y Android, el personal puede supervisar las ventas de automóviles de forma remota desde cualquier lugar. Sus características claves incluyen gestión de inventarios para registrar y revisar el stock de vehículos, ejecución de informes y gestión de costes. Consulta los historiales de los automóviles mediante integraciones con AutoCheck, NMVTIS y Carfax. La función AutoStructure admite acciones administrativas con cálculos de préstamos automatizados y estructuración de acuerdos. Aprovecha las herramientas de gestión de prestamistas de doble canal y la gestión de contratos/documentos para centralizar los documentos administrativos de ventas."

De nuevo, estamos hablando de un sistema con grandes diferencias respecto a la herramienta que se pretende desarrollar en este trabajo, centrándose totalmente en la en la gestión entre un concesionario y sus agentes externos, excluyendo a los clientes. También ofrece un apartado contable; "Proporciona asistencia contable […] junto con las opciones de creación de informes programadas. La adición de los servicios de las agencias de crédito brinda acceso a las verificaciones de crédito del consumidor y facilita el

cumplimiento normativo de cada transacción, con protección contra fraudes y medidas de autenticación integradas. Otras funciones adicionales de DealerCenter abarcan funciones de marketing y promoción de sitios web, las que facilitan la personalización de sitios web de estilo profesional para el concesionario a partir de plantillas y la publicación online de anuncios de vehículos de inventario".

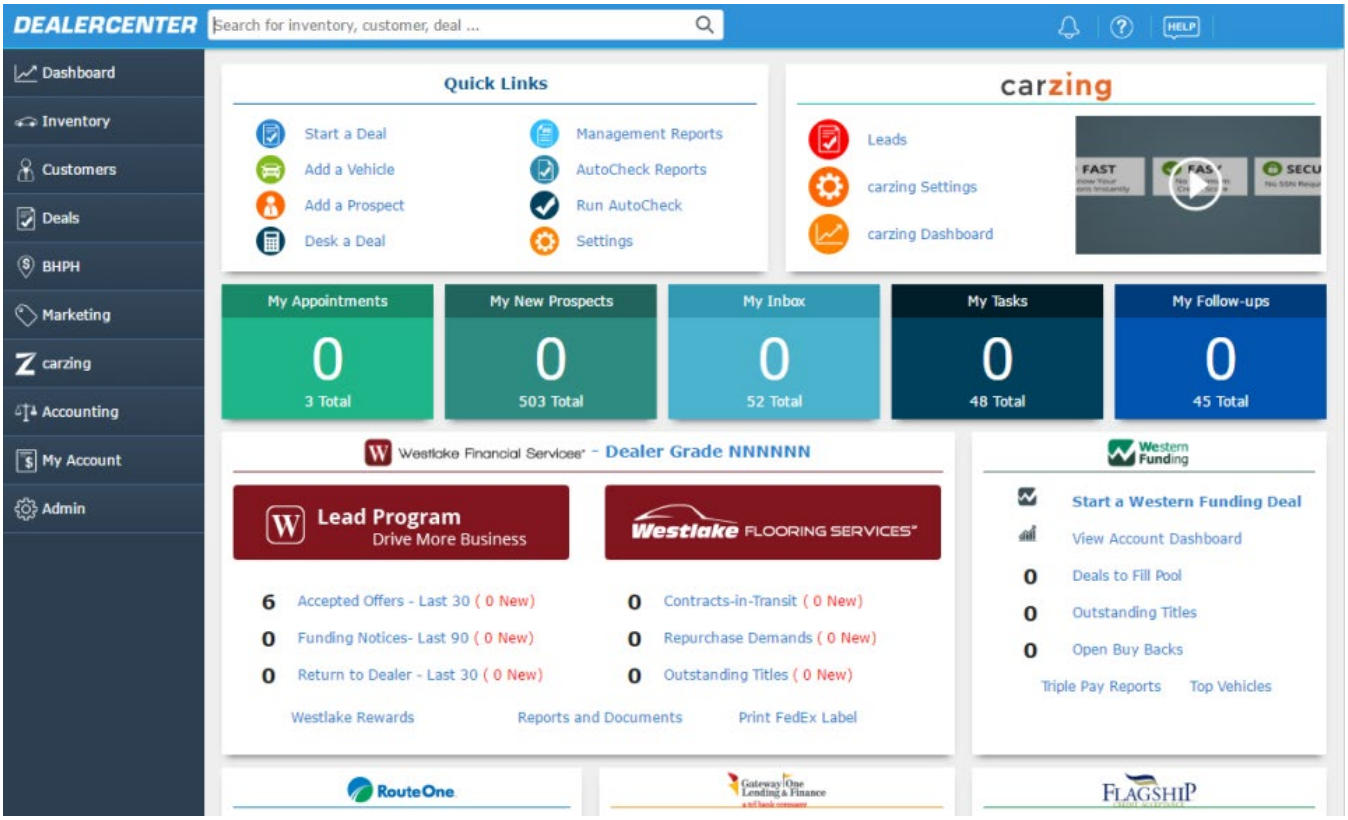


Ilustración 3 - Captura de DealerCenter

2.3 Valor de la solución

Como se puede ver, al ser creado para un cliente concreto con el fin de solventar problemas concretos, nuestro proyecto dispone de forma inherente de todo el mercado necesario para que sea exitoso. Aun así, es posible analizar los puntos fuertes y débiles en caso de que, en un futuro, se pudiera ofrecer a otras empresas que tengan las mismas necesidades.

Por ejemplo, un punto fuerte en la situación de mercado de nuestro proyecto es la posibilidad de crecimiento, tanto horizontal como verticalmente. Por una parte, hay una

gran cantidad de tiendas de compraventa de vehículos y además es un sector que ha crecido en los últimos años, por tanto la lista de clientes potenciales es cada vez mayor.

En el ámbito vertical, también existe la posibilidad de ofrecer más servicios al mismo cliente, tras desplegar la aplicación inicial para los compradores de vehículos, se ofrecerían otras destinadas a ser usadas por los trabajadores del concesionario, ya sea para el control del stock, la contabilidad, la gestión de ofertas e incluso para automatizar la búsqueda de coches para comprar. Convirtiendo un producto en una suite de aplicaciones modulares que cada concesionario contratará en modalidad Software as a Service (SaaS) según sus necesidades.

Otra ventaja de este producto es que la empresa que haga uso de él puede influir en los clientes para que utilicen la aplicación, consiguiendo así un mayor número de *early adopters* en las primeras fases de desarrollo para obtener mejor feedback de las funcionalidades y también facilitar el aumento de usuarios en fases posteriores cuando ya esté el producto completamente implementado y en uso.

Para sintetizar la idea se va a utilizar un "Lean Canvas", un tipo de lienzo de modelo de negocio (también conocido como Business Model Canvas) que es una plantilla gráfica donde se describen las propuestas de producto o de valor para una empresa, además de la infraestructura, dependencias, clientes y finanzas de esta propuesta. Con ello, podremos ver un resumen de cómo se sitúa nuestro proyecto, o al menos una versión simplificada, en el mercado y las aspiraciones que tiene. En algunos casos es útil desarrollar varios Lean Canvas de diferentes productos mínimos viables, para así obtener información de la trayectoria que debería seguir el proyecto y priorizar las diferentes áreas de desarrollo.

La diferencia entre Business Model Canvas y su variación Lean Canvas, es que este último se centra en resolver un problema de los clientes potenciales y es más usado en el ámbito de las nuevas tecnologías.

A partir del Lean Canvas, ya teniendo clara la idea principal sobre la que luego se iterará para mejorar y ofrecer más funcionalidades, podemos empezar a definir requisitos más concretos, comenzar a materializar la futura implementación y centrarse en la obtención de productos mínimos viables que poder entregar para así aportar valor a la empresa de forma continua y escalonada.

PROBLEM	SOLUTION	UNIQUE VALUE PROPOSITION	UNFAIR ADVANTAGE	CUSTOMER SEGMENTS
Los trabajadores del concesionario tienen un tiempo limitado y no pueden atender a más de un cliente a la vez, cuestión que en ocasiones lleva bastante tiempo, lo que genera que otros clientes tengan que esperar en las horas más concurridas, así como también provoca que no todas las llamadas puedan atenderse. El resultado de esto no es solo una disminución de la satisfacción de los clientes si no la pérdida de otros clientes potenciales. **EXISTING ALTERNATIVES** MyDAG	Una vía alternativa, disponible 24h al día en la que los clientes puedan consultar información sin necesitar al personal contratado en el concesionario, así como pedir citas para homogeneizar el número de personas que acuden al concesionario durante el día. **KEY METRICS** Adquisición: Registro en la aplicación Uso: Número de consultas Citas: Citas solicitadas a través del sistema	Una aplicación para diferenciarse de la competencia, aportar más valor percibido para los clientes y automatizar tareas para los empleados **HIGH LEVEL CONCEPT** Siempre disponibles para ayudar al cliente	Los propios vendedores del concesionario podrán recomendar a los clientes y usuarios potenciales que descarguen la aplicación, lo que hace que la adopción sea mucho más amplia. Prácticamente no hay competencia. **CHANNELS** Publicidad en el concesionario Redes sociales Vendedores del concesionario	Clientes del concesionario Personas interesadas en adquirir un vehículo **EARLY ADOPTER** Cualquier cliente del concesionario que utilice su móvil

COST STRUCTURE	REVENUE STREAMS
Desarrollo y mantenimiento de la aplicación Infraestructura y servidores Seguimiento de métricas y estudio de mejoras	Implantación de la aplicación Cobro de la evolución de la aplicación (correctivo y nuevas funcionalidades) Publicidad en la aplicación

Ilustración 4 - Lean Canvas (Maurya, 2014)

2.4 Análisis de requisitos

El primer paso para realizar el análisis de requisitos es la identificación y reconocimiento de los puntos de dolor o problemas que los stakeholders puedan tener, para esto se han realizado varias entrevistas con los empleados y clientes del concesionario para que detallen los procesos de negocio recurrentes en su jornada o las necesidades que no veían satisfechas, de estas entrevistas se han podido identificar, entre otros, los siguientes problemas y sus consecuencias potenciales.

Problema 1: Tiempo de espera de los clientes

En las horas más concurridas, es probable que los clientes del concesionario tengan que esperar mucho tiempo para ser atendidos, ya que los vendedores están ocupados con otros clientes o llamadas.

Consecuencias:

- Pérdida de clientes potenciales
- Disminución de la satisfacción del cliente
- Peor percepción de la marca
- Perdida de cercanía y profesionalidad en el trato

Problema 2: Archivado de documentos físicos

Todas las compras, ventas y traspasos de vehículos generan una serie de documentos que se almacenan físicamente en varios archivadores, agrupados en carpetas por vehículo.

Consecuencias:

- Posible pérdida de documentos al guardarlos incorrectamente
- Alto tiempo de acceso y archivo
- Dificultad para enviar una copia a los clientes
- Necesidad de un espacio físico para guardarlos
- Acumulación con el tiempo
- Falta de acceso desde otros puestos de venta

Problema 3: Incapacidad de contactar con los vendedores por teléfono

El primer problema también causa que los vendedores no puedan atender llamadas ya que suelen priorizar a los clientes que se encuentran físicamente en el concesionario, por tanto, el teléfono suele recibir multitud de llamadas que no son atendidas.

Consecuencias:

- Mismas consecuencias que el Problema 1
- Molestias para todas las personas presentes en el concesionario
- Frustración de los clientes que llaman por teléfono
- Necesidad de desplazarse a la tienda para cualquier consulta

Una vez se hayan juntado requisitos concretos que solicitaban los empleados, clientes y propietarios del concesionario con soluciones que se les han propuesto a los mismos tras analizar sus flujos de trabajo, se puede decir que ya se han obtenido las necesidades que debe cubrir este proyecto, y por tanto se les propone una posible solución además de un plan detallado de cómo llegar a ella partiendo de la situación actual.

2.5 Diseño conceptual de la solución

En este caso concreto, se ha diseñado una herramienta específica que permite atacar todos los problemas al mismo tiempo, además de conseguir mejorar la eficiencia del personal del concesionario, posibilitando que atiendan a un mayor número de clientes en el mismo tiempo y proveer a la empresa de una herramienta de diferenciación frente a la competencia.

Más concretamente, la solución propuesta consta de 3 partes principales.

El primer paso es la creación de una base de datos donde se pueda guardar tanto la información de los vehículos y clientes como de las propias transacciones y toda la documentación que estas conllevan. Alojada en un servidor remoto, así los empleados de cualquier concesionario de la franquicia podrían acceder a la información desde el ordenador del que disponen para trabajar.

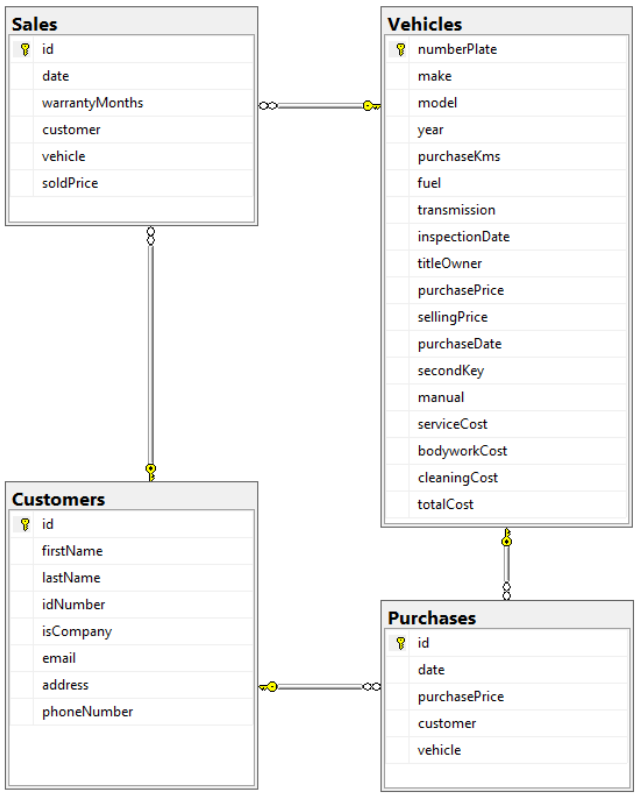


Ilustración 5 - Esquema de datos inicial

Es evidente que esto apelaría al problema del archivado de documentos físicos y aunque es cierto, también es la base para que el resto de las aplicaciones funcionen correctamente, nutriéndolas de datos.

Otra de estas aplicaciones mencionadas es una app móvil, en principio diseñada para Android, con la que los clientes pueden obtener información para la que normalmente tendrían que llamar o personarse en un concesionario. Además de estar destinada a solventar los problemas de tiempo de espera de los clientes y del alto volumen de llamadas hacia los vendedores, esta aplicación también ofrecerá un nuevo canal de comunicación con las personas que ya hayan adquirido un vehículo o estén pensando en hacerlo.

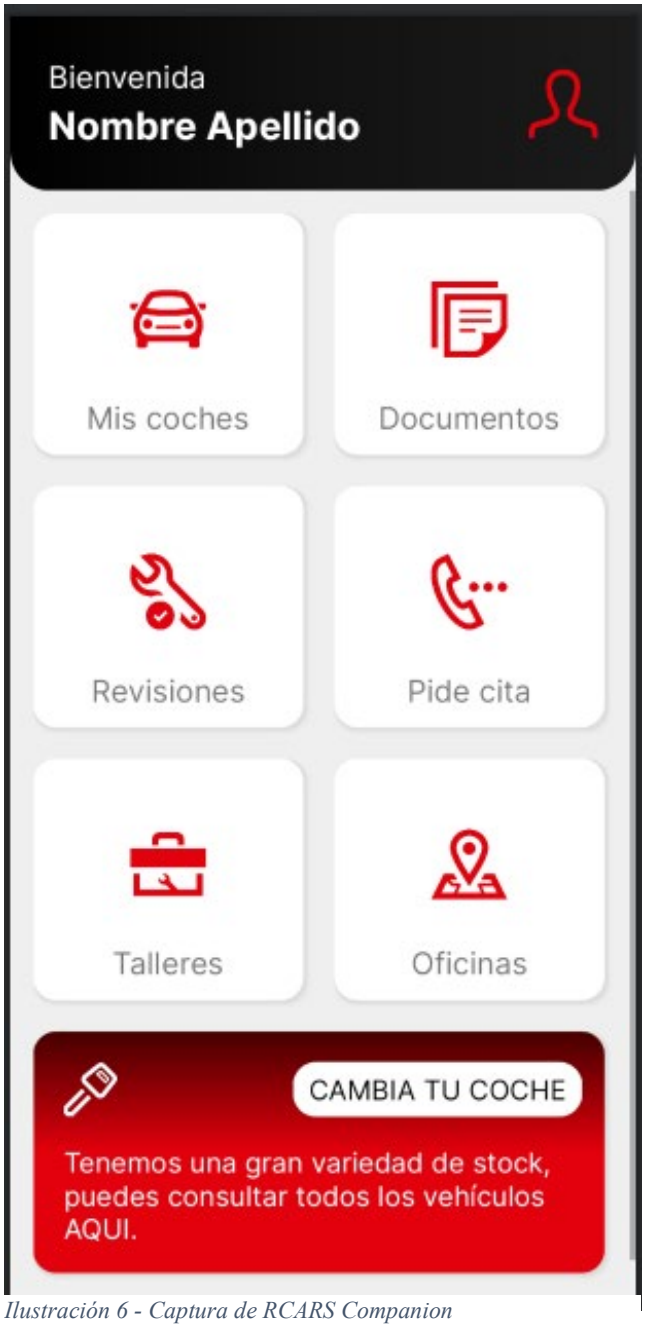


Ilustración 6 - Captura de RCARS Companion

Las principales funciones de esta app, antes de acordarse los requisitos concretos, eran que los clientes pudiesen acceder a la información detallada de los vehículos que han adquirido, así como a toda la documentación que los concierna. Además, los usuarios deberían poder ver los coches actualmente en venta, así como información de los concesionarios como la ubicación, los teléfonos, etc.

Por último, se implementará otra pieza de software que actuará de intermediario entre la base de datos y la aplicación móvil (y otras posibles aplicaciones). Se usará una API, que como su propio nombre indica (Application Programming Interfaces) es una interfaz que recibe peticiones y responde con información. Se podría definir como el contrato de servicio entre múltiples aplicaciones.

Esta API en concreto, desarrollada en .NET, se encargará de gestionar la persistencia en la base de datos, así como la obtención de la información de la misma, incluyendo también de forma inherente lógica de negocio y control de acceso.

Se alojará en un servidor de Azure donde la app móvil podrá conectarse y realizar consultas tipo REST para obtener, por ejemplo, una lista con todos los vehículos de un cliente. (ULPGC)

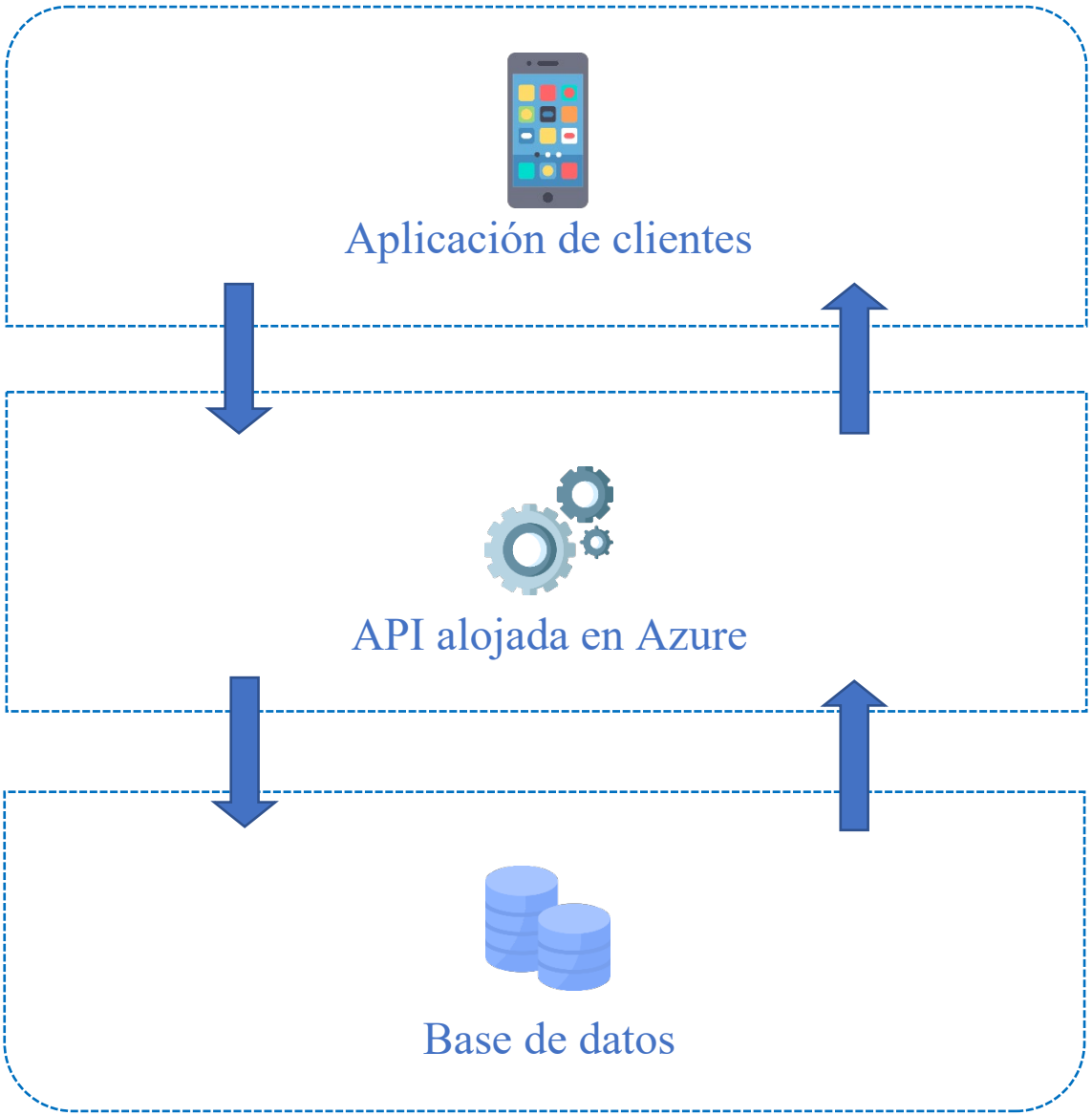


Ilustración 7 - Esquema de la arquitectura

2.6 Especificación de requisitos

Tras acordar unas bases superficiales de lo que será el proyecto, se ha procedido a definir unos requisitos concretos con el Product Owner. Hay que decir que estos requisitos no son lo suficientemente específicos como se esperaría en un desarrollo normal, esto se debe a que por parte de RCARS dejaban en manos del desarrollador los detalles, que se especificaron en una fase más tardía de la implementación. Por otra parte, los requisitos han sufrido ligeros cambios a lo largo del proyecto, quedando algunos deprecados en favor de modificaciones futuras solicitadas tras cambiar de idea o tras probar una versión preliminar del sistema. (ULPGC)

Para la correcta priorización de los primeros requisitos, así como para agruparlos más fácilmente en productos entregables, se ha utilizado la técnica conocida como Story Mapping, que como explica en el libro homónimo su creador Jeff Patton, trata de mantener siempre la entrega de valor centrada en las necesidades del cliente construyendo modelos simples.

Esto se consigue creando una cuadrícula con todas las tareas que el usuario del producto o servicio requiere para así trazar un recorrido de posibles entregas calculadas a partir del valor que recibiría el cliente.

A continuación se muestra la cuadrícula anteriormente mencionada y además se recogen algunos de los principales requisitos funcionales acordados con los stakeholders con respecto a la app móvil:

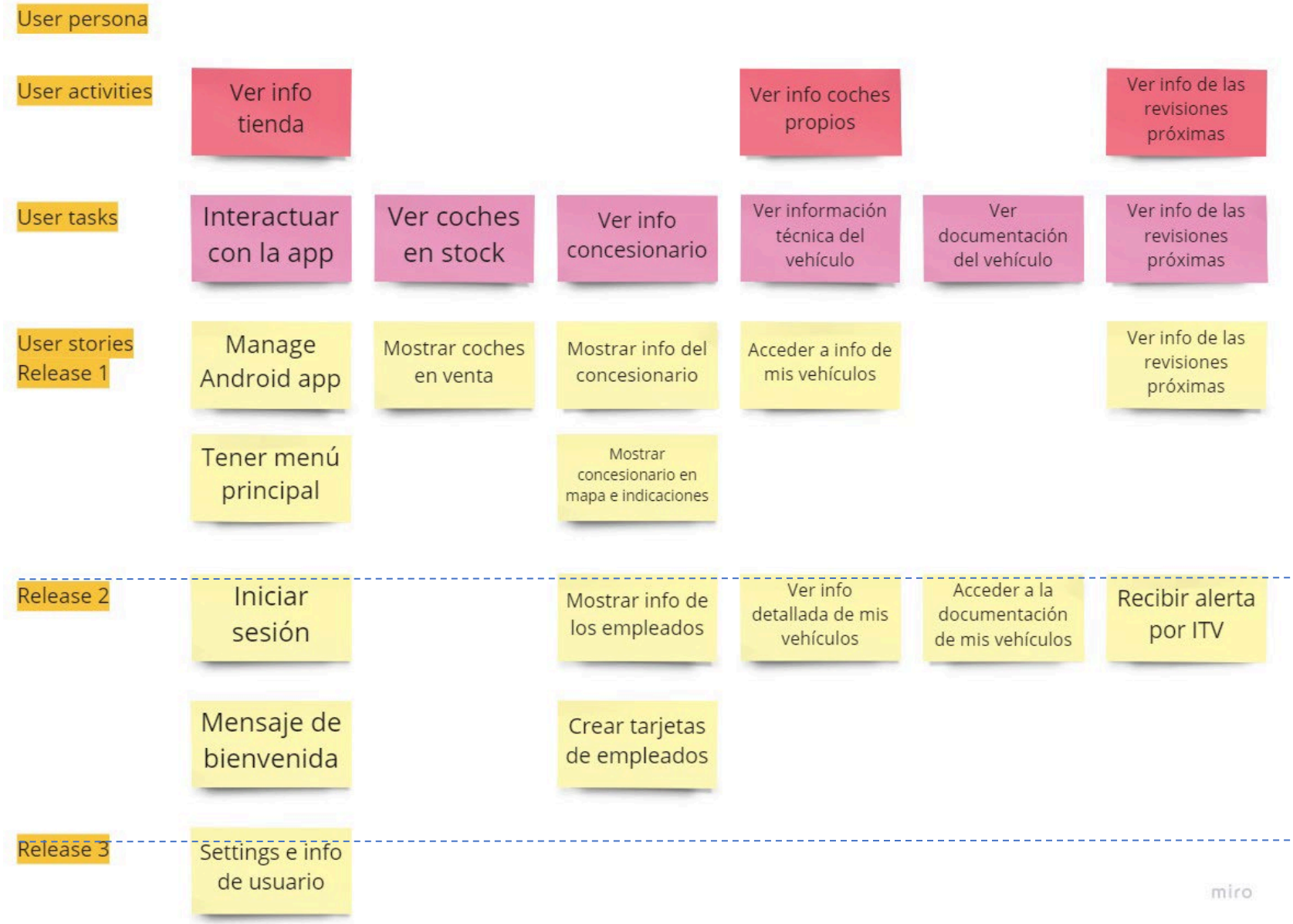


Ilustración 8 - User Story Mapping (Patton, 2015)

Poder iniciar sesión	DEV-01
DESCRIPCIÓN	
COMO usuario de la app QUIERO poder iniciar sesión PARA acceder a toma mi información	
CRITERIOS DE ACEPTACIÓN	
– Requerir correo y contraseña – Mostrarse al abrir la app – Comprobar credenciales en el Backend	
ASPECTOS FUNCIONALES	
– **UI/UX-Descripción de los cambios funcionales realizados** Diseño simple – **Casos de uso** Los usuarios ya deben estar creados por nuestros trabajadores	
ASPECTOS TÉCNICOS	
– **Tablas implicadas** Users – **Campos a validar** Email, password	
OBSERVACIONES/NOTAS	

Mostrar coches en venta	DEV-02
DESCRIPCIÓN	
COMO usuario de la app QUIERO ver los coches en stock de los concesionarios PARA saber que hay disponible para comprar	
CRITERIOS DE ACEPTACIÓN	
Los usuarios deben poder acceder desde el menú principal a la página de RCARS	
ASPECTOS FUNCIONALES	
– **UI/UX - Descripción de los cambios funcionales realizados** El acceso a esta parte debe ser diferente a las demás	
ASPECTOS TÉCNICOS	
– **Configuración** Enlace : https://www.coches.net/concesionario/rcars/	
OBSERVACIONES/NOTAS	
Redirigir a la web de la tienda, al volver para atrás tiene que volver a la app.	

Acceder a información de mis vehículos	DEV-04
DESCRIPCIÓN	
COMO usuario de la app que ha adquirido un vehículo QUIERO acceder a la información técnica de mis coches PARA poder conocer los detalles de este	
CRITERIOS DE ACEPTACIÓN	
– Un usuario tiene que ver un listado de todos los vehículos que ha comprado – Mostrar logo de la marca, marca, modelo y año – Si no tiene ningún vehículo mostrar acceso al concesionario online	
ASPECTOS FUNCIONALES	
– **UI/UX-Descripción de los cambios funcionales realizados** Acceder desde el menú principal	
ASPECTOS TÉCNICOS	
– **Descripción de los cambios técnicos realizados** Obtener listado de un nuevo endpoint de la API, respetando RESTful – **Tablas implicadas** Vehicles, Customers, Sales – **Campos a validar** CustomerId, NumberPlate	
OBSERVACIONES/NOTAS	
Logos proporcionados por RCARS	

Mostrar mensaje de bienvenida	DEV-06
DESCRIPCIÓN	
COMO dueño del sistema QUIERO que los usuarios vean un mensaje de bienvenida personalizado PARA que reciban un trato personal	
CRITERIOS DE ACEPTACIÓN	
– Mostrar mensaje de bienvenida en la página principal [DEV-05] – Debe incluir el nombre del usuario/a	
ASPECTOS FUNCIONALES	
– **UI/UX-Descripción de los cambios funcionales realizados** Situarlo fijo, en la parte superior del menú principal	
ASPECTOS TÉCNICOS	
– **Tablas implicadas** Users, Customers	

Acceder a información de mis vehículos en detalle	DEV-08

DESCRIPCIÓN

COMO usuario de la app que ha adquirido un vehículo
QUIERO acceder a la información técnica de mis coches
PARA poder conocer los detalles de este

CRITERIOS DE ACEPTACIÓN

- Desde la vista de listado de mis vehículos [DEV-04] poder acceder a los detalles de un vehículo concreto
- Mostrar toda la información disponible de ese vehículo menos la documentación y la referente al gasto de la empresa.
- Mostrar acceso a la documentación del vehículo [DEV-09]
- Esconder el resto de los vehículos de la lista
- Al volver atrás, mostrar otra vez el listado de vehículos

ASPECTOS FUNCIONALES

- **UI/UX-Descripción de los cambios funcionales realizados**
  Acceder al seleccionar un vehículo de la lista
  Mostrar la información en un solo bloque, como un documento

ASPECTOS TÉCNICOS

- **Descripción de los cambios técnicos realizados**
  Obtener info completa de un endpoint de la API diferente, para que no cargue toda la info de todos los coches al entrar al listado
- **Tablas implicadas**
  Vehicles, Customers, Sales
- **Campos a validar**
  CustomerId, NumberPlate

OBSERVACIONES/NOTAS

Acceder a la documentación de mis vehículos	DEV-09

DESCRIPCIÓN

COMO usuario de la app que ha adquirido un vehículo
QUIERO acceder a la documentación de mis coches
PARA poder hacer cualquier tramite

CRITERIOS DE ACEPTACIÓN

- Desde el menú principal, poder acceder a un apartado de "Documentos"
- Mostrar toda la documentación disponible de ese cliente.
  [Contrato de compraventa, traspaso, protección de datos, documentación del vehículo]
- Permitir descargar documentos [DEV-10]
- Documentos subidos previamente por empleados de RCARS [DEV-11]

ASPECTOS FUNCIONALES

- **UI/UX-Descripción de los cambios funcionales realizados**
  Mostrar título de los documentos
  Abrirlos al pulsar sobre ellos

ASPECTOS TÉCNICOS

- **Tablas implicadas**
  Documents, Customers
- **Campos a validar**
  CustomerId

Ver información de trabajadores	DEV-12

DESCRIPCIÓN

COMO usuario de la app
QUIERO ver información de los trabajadores del concesionario
PARA poder contactar con ellos

CRITERIOS DE ACEPTACIÓN

- Desde el menú principal, poder acceder a un apartado de "Pedir cita"
- Mostrar nombre, foto, teléfono y horario de cada empleado y el general de la tienda

ASPECTOS FUNCIONALES

- **UI/UX-Descripción de los cambios funcionales realizados**
  Usar tarjetas creadas por la empresa

ASPECTOS TÉCNICOS

- **Parametrización**
  Posibilidad de almacenar esta info en un futuro y no *hardcodearla*

OBSERVACIONES/NOTAS

Mostrar información de los concesionarios	DEV-15

DESCRIPCIÓN

COMO usuario de la app
QUIERO saber dónde está situado el concesionario
PARA poder ir en cao de que lo necesite

CRITERIOS DE ACEPTACIÓN

- Desde el menú principal, poder acceder a un apartado de “Concesionarios”
- Mostrar la posición de los concesionarios en el mapa
- Permitir mostrar indicaciones de cómo llegar desde la posición actual

ASPECTOS FUNCIONALES

- **UI/UX-Descripción de los cambios funcionales realizados**
- **Capturas de pantalla (añadir en el momento)**

ASPECTOS TÉCNICOS

- **Descripción de los cambios técnicos realizados**
  Usar API Google Maps
- **Tablas implicadas**
  Dealers

Mostrar información de próximas revisiones	DEV-16

DESCRIPCIÓN

COMO usuario de la app que ha adquirido un vehículo
QUIERO ver las próximas revisiones y mantenimiento estipuladas
PARA saber cuándo tengo que llevarlo al taller

CRITERIOS DE ACEPTACIÓN

- Desde el menú principal, poder acceder a un apartado de “Revisiones”
- Mostrar , separado por vehículo, las revisiones y mantenimiento programadas
- Incluir la próxima ITV

ASPECTOS FUNCIONALES

- **UI/UX-Descripción de los cambios funcionales realizados**
  Mostrar todos los vehículos del cliente, aunque no tengan revisiones pendientes

ASPECTOS TÉCNICOS

- **Tablas implicadas**
  Vehicles

OBSERVACIONES/NOTAS

Info para calcular la próxima ITV - pdf/2017/BOE-A-2017-12841-consolidado.pdf

Tener menú principal	DEV-05
DESCRIPCIÓN	
COMO usuario de la app QUIERO poder acceder a un menú principal PARA ver las acciones que puedo realizar	
CRITERIOS DE ACEPTACIÓN	
– Acceder tras iniciar sesión – Incluir acceso a principales funciones de la app [DEV-04] – Destacar acceso al concesionario online [DEV-02]	
ASPECTOS FUNCIONALES	
– **UI/UX-Descripción de los cambios funcionales realizados** Diseño minimalista Fácil visibilidad – **Casos de uso** Añadir nuevos casos con futuras historias – **Ejemplos funcionales** App móvil Banco Santander	

Generar alerta ITV	DEV-21
DESCRIPCIÓN	
COMO usuario de la app que ha adquirido un vehículo QUIERO que se me avise cuando necesite pasar la ITV PARA poder saberlo con tiempo	
CRITERIOS DE ACEPTACIÓN	
– Notificar al usuario cuando uno de sus vehículos necesite pasar la ITV en menos de un mes – Notificar al usuario cuando uno de sus vehículos necesite pasar la ITV en menos de 3 días	
ASPECTOS FUNCIONALES	
– **UI/UX-Descripción de los cambios funcionales realizados** – Notificación externa a la app	
ASPECTOS TÉCNICOS	
– **Descripción de los cambios técnicos realizados** – **Tablas implicadas** Vehicles – **Campos a validar** InspectionDate	
OBSERVACIONES/NOTAS	
La mayoría de los coches vendidos no han pasado la primera ITV	

2.7 Tecnologías consideradas

En un principio se consideró la realización de la aplicación móvil mediante React Native, este framework de JavaScript es extremadamente popular, por lo que existe mucha documentación e información en internet en el caso de necesitarlo.

Una de las principales ventajas es que se podría realizar una aplicación móvil multiplataforma, en lugar de crear dos versiones, una para Android y otra para iOS. Usando el mismo código para ambas plataformas significa reducir los costes tanto de tiempo como de dinero significativamente, tanto en el desarrollo como en el mantenimiento.

Sin embargo, para depurar una aplicación desarrollada para iOS, es necesario un equipo con MacOS, como un Mac o MacBook. Eso hizo que la decisión final fuera realizar la aplicación en Android Studio, ya que también existe una comunidad extensa y activa de desarrolladores e información disponible y al final el resultado iba a ser el mismo, y lo que buscaba RCARS para empezar, una app solo para dispositivos Android.

A pesar de no ser una ventaja inherente, otro de los principales motivos de la elección de Android Studio para el desarrollo se debe a que en el grado se enseña mucho más Java que JavaScript y por tanto el nivel de conocimiento es mucho mayor y esto hará que cueste menos la implementación.

Por otro lado, no hay que olvidar los beneficios de realizar la aplicación con una tecnología nativa, ya que así se consigue un mayor rendimiento en comparación con una app híbrida o multiplataforma, aprovechando las bondades del lenguaje de programación ya que el sistema operativo sobre el que funcionará está diseñado para sacarle todo el partido posible, obteniendo así una app optimizada para cualquier teléfono móvil Android.

Además del rendimiento, la experiencia de usuario también será considerablemente mejor, pudiendo desarrollar la capa de presentación usando herramientas que generan elementos visuales característicos de Android, ofreciendo unas vistas y usos a los que el usuario ya está acostumbrado y con los que se maneja con soltura.

Para comunicarnos con la API, desarrollada en .NET, se ha usado Retrofit. Esta librería consigue, de forma resumida, convertir una API HTTP en una interfaz de Java, por tanto, con muy poco código podemos generar llamadas a la API como si estuviéramos invocando un método más de nuestro código y esto, sumado a GSON que es una herramienta desarrollada por Google para convertir código JSON a objetos Java (deserialización) o viceversa (serialización), hace que la comunicación entre la API y la app móvil sea muy sencilla.

Como se ha mencionado previamente, la API se ha desarrollado utilizando .Net 6 y C# 10, se han utilizado estas versiones en concreto debido a que ofrecen una característica que antes no existía, las llamadas “Minimal APIs”.

Estas interfaces ofrecen el mismo funcionamiento que cualquier otra, sin embargo, son mucho más fáciles de implementar y extremadamente más sencillas, con una cantidad mínima de código *boilerplate* (secciones de código que se repiten en multitud de partes del código con variaciones nulas o mínimas), haciéndolas ligeras y fáciles de modificar, compilar y ejecutar. De esta tecnología, así como del IDE Visual Studio, hablaremos más adelante.

Otro factor importante en la elección de .Net, es el planteamiento futuro del sistema, ya que se pretende diseñar otra aplicación web para el uso de los empleados del concesionario, además resulta importante la experiencia adquirida en el ámbito profesional, ya que en las prácticas externas de la universidad se ha trabajado casi exclusivamente con el stack tecnológico de Microsoft, obteniendo experiencia con el lenguaje concreto y con el entorno de desarrollo, así como también del portal de Azure que, perteneciendo también a Microsoft, pone a disposición del desarrollador el uso de sus herramientas consiguiendo así facilitar la integración y el despliegue de la API, el seguimiento con Application Insights, la integración continúa con la creación de pipelines, etc.

Además, en caso de necesitarlo, gracias a la licencia universitaria se puede utilizar de forma gratuita Rider, el entorno de desarrollo de JetBrains para C# famoso por su rendimiento y por ReSharper, una herramienta de refactorización muy útil con la que se consigue de forma sencilla un código limpio.

La interacción entre la API y la base de datos se ha hecho mediante la utilización de un ORM (Object-Relational Mapping), es un mecanismo para mapear tablas o entradas de una base de datos a un POJO (Plain Old Java Object) o en nuestro caso, un objeto de C#. La comunicación es bidireccional por tanto también es posible editar una instancia de una clase de nuestro código y guardarla directamente en la base de datos, siendo el ORM, concretamente Entity Framework en nuestro caso, el que se encarga de la transformación necesaria para persistir el objeto en una tabla SQL. Por todo ello, las operaciones CRUD de nuestro sistema son transparentes a la implementación del sistema de escritura o lectura en la base de datos.

Para terminar, la base de datos por la que se ha optado es una clásica base de datos relacional SQL (Con SQL Server) por la simplicidad de uso, la popularidad del sistema y la robustez demostrada durante todo el tiempo que lleva en el mercado.

Con la simple creación de un servidor que aloje esta base de datos y teniendo un servicio activo, cualquier aplicación que disponga de la cadena de conexión podrá acceder a la información contenida en la base de datos.

3 Diseño

3.1 Arquitectura del sistema

Para este sistema se ha optado por una arquitectura por capas, este tipo de diseño es uno de los más utilizados debido a su simpleza y el poco riesgo en cuanto a ser una mala elección. Consta en dividir una aplicación en diferentes niveles y asignarle un rol distinto a cada uno, lo más usual es separar la aplicación en capa de presentación, de negocio, de persistencia y de datos. Sin embargo, esta arquitectura no define cuantas capas concretas debe presentar la aplicación, lo importante es la separación de intereses o SoC por sus siglas en inglés (Separation of Concerns) por la cual cada capa debería tener una responsabilidad única concreta. (Pérez) (Hürsch & Lopes, 1995)

En nuestro caso concreto, la capa de negocio y persistencia están prácticamente unidas, esto se debe a que dentro del negocio se utiliza un ORM (Entity Framework) que facilita tanto el acceso a los datos que no resulta necesario tener una capa dedicada a ello.

El término "capa" se utiliza en este ámbito debido a que, si representamos gráficamente todas estas "partes" de la aplicación como capas una encima de la otra, resulta intuitivo entender que cada una puede comunicarse solamente con su inmediata superior e inferior, como se ve en la siguiente imagen que describe nuestro sistema y donde las flechas azules representan *request* o peticiones y las naranjas respuestas.

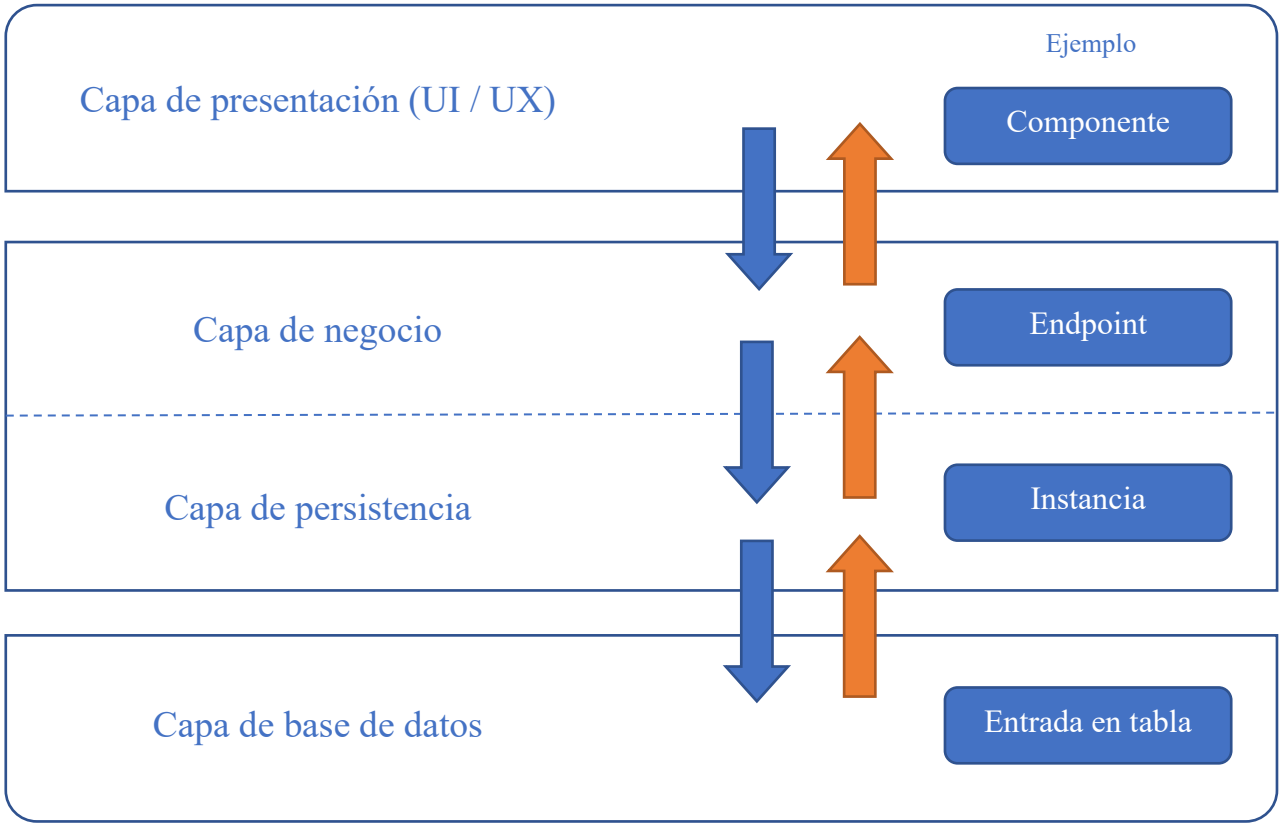


Ilustración 9 - Esquema de capas

Tras ver la imagen queda claro que si una capa superior como la de presentación quiere comunicarse con la de datos debe hacerlo mediante las capas intermedias, a base de envío de peticiones y recepción de respuestas.

Al estar estructurado de esta manera y si, siguiendo las normas arquitectónicas de esta metodología, cada capa es un componente distinto, podríamos desplegar un componente sin afectar a los demás, alojarlos en diferentes servidores e incluso cambiar una implementación de una capa por otra. Por ejemplo, si ya no nos interesa que la capa de presentación este hecha en React y optamos por una aplicación nativa Android, mientras cumpla el contrato de peticiones y respuestas el resto de las capas no tienen por qué verse afectadas con este cambio, de hecho, para estas es indiferente esa decisión.

Lo que se busca con la separación de las capas es como se mencionó anteriormente es la separación de intereses, es decir, que cada capa se encargue de una tarea concreta y definida, por ejemplo, la capa de negocio no tiene por qué saber cómo se mostrará la información al usuario, simplemente ha de ofrecer un json, xml u otro tipo de archivo y la capa de presentación se encargará de transformarlo para mostrarlo de una forma más amigable al usuario. A la inversa, a la capa de presentación no le importa de donde viene la información o la lógica del negocio, no realiza comprobaciones de ese tipo cuya labor pertenece a la capa de negocio.

Así mismo, la capa de negocio o persistencia simplemente ordena que cierto dato se guarde, no conoce si se guardará en una base de datos, en un log, en un fichero de texto, etc. Ese trabajo pertenece a la capa de datos y si en un futuro cambia no debería implicar ninguna adaptación en la capa superior. De hecho, sería posible implementar dos capas de datos diferentes a la vez, una guarda los datos en una base de datos relacional SQL y la otra en MongoDb y la capa de persistencia sería la misma para las dos.

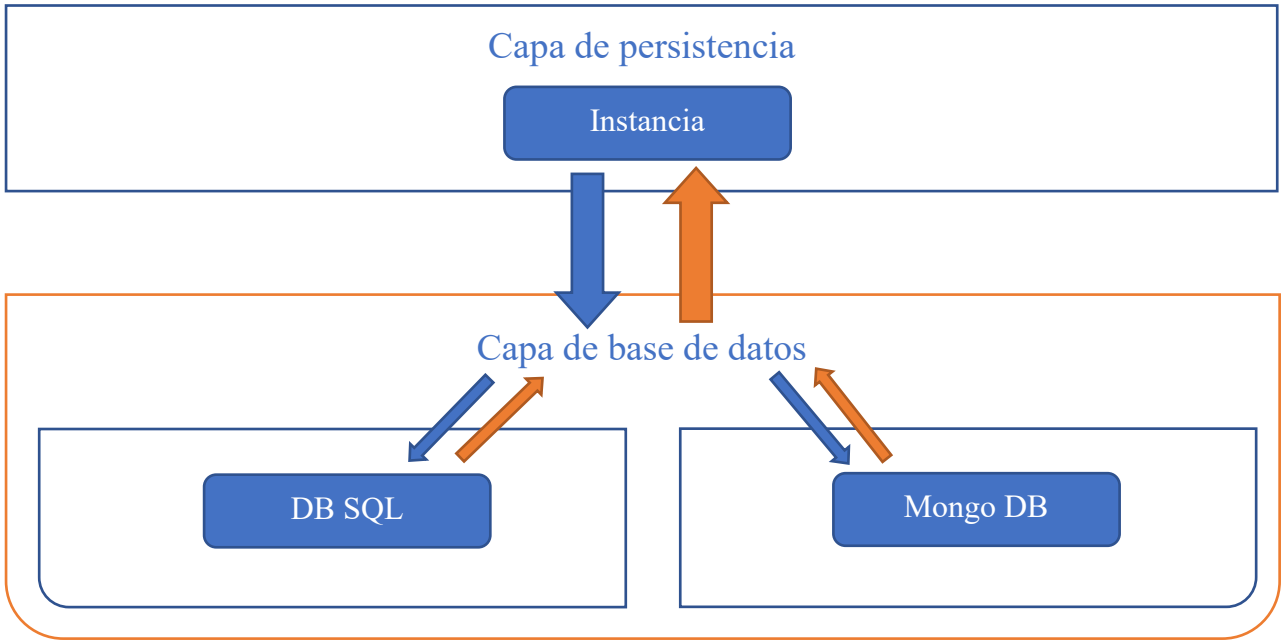


Ilustración 10 - Esquema de capas inferiores

No hay ningún impedimento físico o lógico en que capas no contiguas se comuniquen, esto hace que haya que tener mucho cuidado ya que si no saltamos alguna capa en la transmisión de una petición, el código seguirá compilando e incluso puede parecer en primeras etapas del desarrollo más simple y limpio, sin embargo, esto puede generar a la larga dependencias que impliquen una dificultad mucho mayor al hacer cambios en el futuro. Si se pierden la modularidad, la separación de intereses y las dependencias en capas, al final se genera un anti-patrón, también conocido como código espagueti, donde cada modificación en el sistema será extremadamente compleja y el código resultará difícil de leer.

El objetivo es sentar las bases para, en una fase posterior con más aplicaciones conectadas, poder realizar una arquitectura hexagonal de una forma sencilla.

En conclusión, las principales ventajas son la separación de responsabilidades, la facilidad de desarrollo y mantenimiento y otra que no se ha nombrado antes, la facilidad de realizar pruebas, ya que teniendo las responsabilidades separadas se pueden generar dobles, mocks o stubs de una forma sencilla para testear el resto del sistema.

Obviamente también tiene ciertas desventajas como, por ejemplo, el mantenimiento de la comunicación entre las diferentes capas ya que si una capa falla no existe comunicación en el sistema.

3.2 Diseño del dominio

El dominio de la información resulta fácil de entender al formarse por entidades y relaciones a las que estamos acostumbrados, como que un cliente compra un vehículo, sin embargo, hay que especificar de forma muy concreta tanto las clases y sus atributos como las relaciones entre ellas. (ULPGC)

Aquí resulta de gran ayuda hablar con la gente que más control tiene sobre el negocio y sus procesos, los trabajadores. Son ellos los que mejor conocen los datos necesarios que tienen que transmitirse entre las diferentes entidades, así como las relaciones entre las mismas.

Para concretar las entidades que se deben implementar en el sistema, lo que se conoce como dominio de la información o del negocio, se ha descrito el siguiente esquema, obviando las propiedades de cada elemento puesto que se explicarán más adelante en el esquema de datos y siendo aquí, solo un subconjunto de estas.

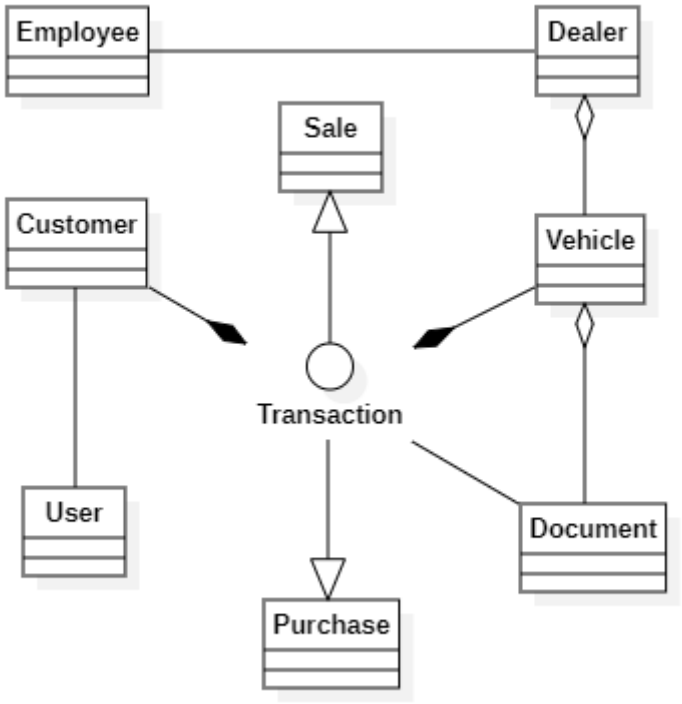


Ilustración 11 - Esquema del dominio del negocio

3.3 Prototipado

Para realizar el prototipado de la interfaz gráfica del usuario, se ha utilizado la herramienta Figma. Con ella se han fabricado capturas, transiciones y animaciones para consultar con el product owner con el fin de que pueda dar su aprobación antes de la implementación y no después, dado que realizar un cambio llevaría más esfuerzo y tiempo.

A continuación se mostrarán capturas del prototipo realizado tras varias reuniones con los stakeholders en las que sufrió sendos cambios. Tras estas iteraciones se consiguió que se

Ilustración 12 - Captura de pantalla de inicio de RCARS Companion

considerase como suficientemente bueno, a pesar de que en el resultado final se aprecien algunas diferencias que se exigieron más adelante.

Además de ver como serían las vistas de la aplicación, el prototipado también puede mostrar comportamiento, esto se conoce como *wireframe* y enseña cómo se desenvolvería la aplicación tras interactuar con el usuario, por ejemplo, al tocar un botón.

El diseño de un prototipo también puede descubrir al product owner nuevos requisitos en los que no se habían percatado. Al ver la app "funcionando" puede uno darse cuenta de cierta funcionalidad que había dado por hecho y no había mencionado o descubrir una necesidad nueva.

En resumen, el prototipado sirve para poner en consonancia a los stakeholders involucrados en el desarrollo, facilitando que estos refinen detalles de los requisitos y dando una imagen a algo hasta ahora abstracto.

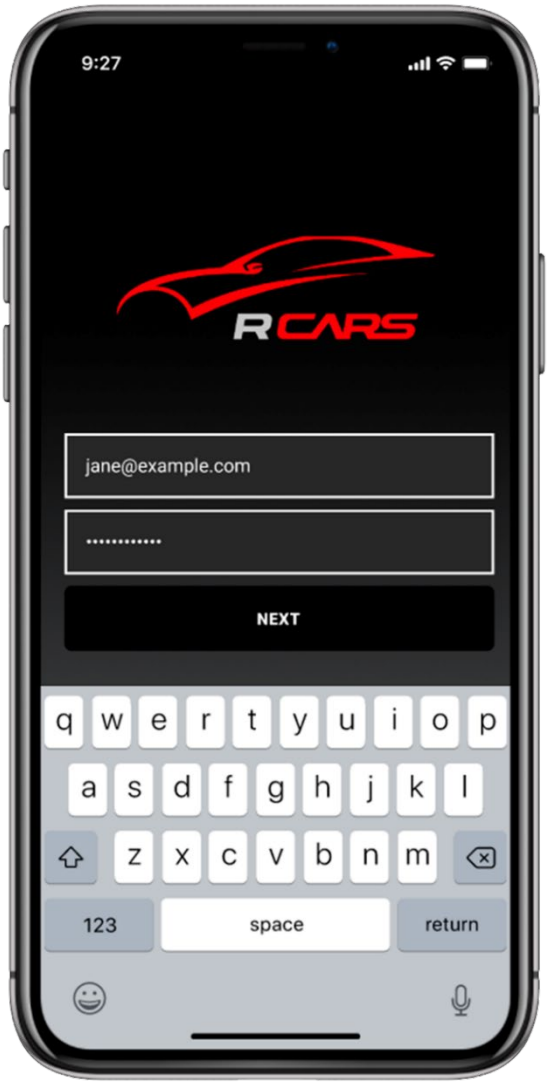


Ilustración 14 - Captura de pantalla de inicio de sesión

Ilustración 13 - Captura de pantalla de menú principal

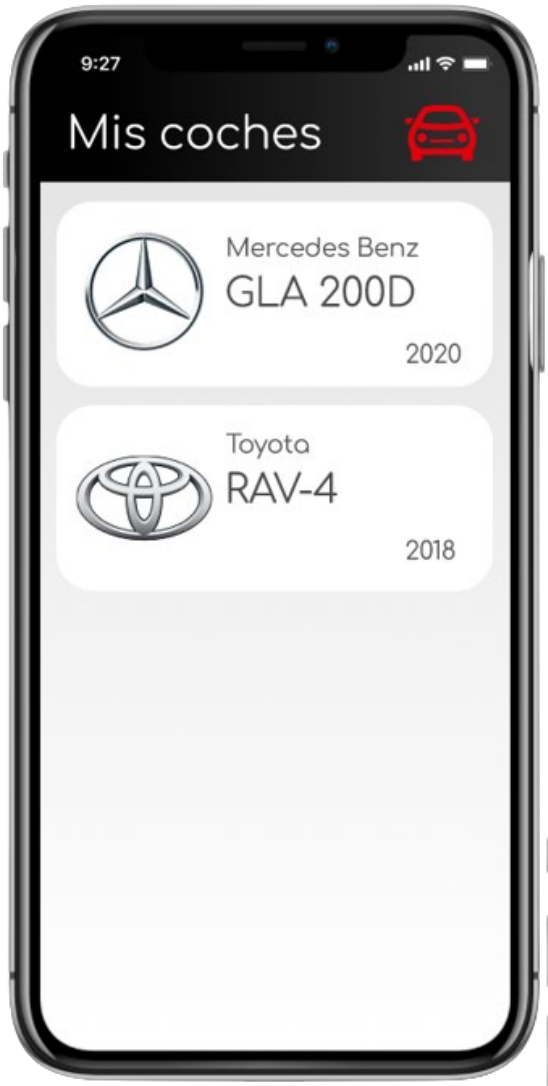


Ilustración 16 - Captura de pantalla de la sección "vehículos"

Ilustración 15 - Captura de pantalla del detalle de un vehículo

Una de las características principales de este diseño, que se acaparó gran parte del tiempo de reunión, es la facilidad de uso. Todas las funciones principales están visibles desde el menú principal tras iniciar sesión, ninguna de ellas se encuentra tras un desplegable o dentro de un submenú y puesto que los usuarios potenciales de esta aplicación es gente de todas las edades y que están acostumbradas a utilizar el móvil en mayor o menor medida, resulta muy intuitiva para su uso.

3.4 Entidades y persistencia

Para las primeras versiones del sistema, aquellas MVP cuya complejidad no es muy grande, no es necesario almacenar en base de datos una gran cantidad de entidades diferentes ni que estas contengan muchos campos. Con el tiempo, estas tablas irán creciendo y multiplicando a medida que se expanda la usabilidad del sistema.

En este punto, las entidades con las que trabaja la aplicación mayormente son vehículos (Vehicles), clientes (Customers) que tienen asociado un usuario (User), concesionarios (Dealers) y las posibles relaciones que hay entre ellos como Ventas (Sales) refiriéndose a la venta de un vehículo a un cliente o Compras (Purchases) en el caso inverso.

Cualquiera de estas entidades puede recorrer todas las capas de la aplicación, por lo que cada una tiene entiende una implementación de la misma. Hay una clase Vehicle, de Java, en la aplicación móvil, cuya instancia concreta necesita actualizarse en la base de datos. Para ello hay otra clase, esta vez en C#, en la API que mediante serializaciones y deserializaciones de objetos JSON, puede recibir o enviar la información de la instancia concreta. De este estándar de comunicación hablaremos más adelante.

La comunicación y transferencia de entidades entre las dos capas más inferiores es diferente, de esto se ocupa una tecnología desarrollada por Microsoft, Entity Framework.

Como se ha explicado brevemente en el capítulo anterior, un asignador relacional de objetos, esto significa que permite trabajar con tablas de una base de datos como si se tratase de objetos propios del sistema y a la vez hace prescindible gran parte del código necesario para acceder y convertir los datos alojados fuera de la aplicación.

Por último, la base de datos es agnóstica a todas estas tecnologías, transferencias y protocolos. Esta está alojada en un servidor y simplemente almacena la información en tablas relacionadas entre sí. Tanto para acceder a los datos como para modificarlos, eliminarlos o cambiar su estructura, hacen falta sentencias SQL.

A continuación se puede ver cómo están estructuradas las diferentes entidades a nivel de base de datos.

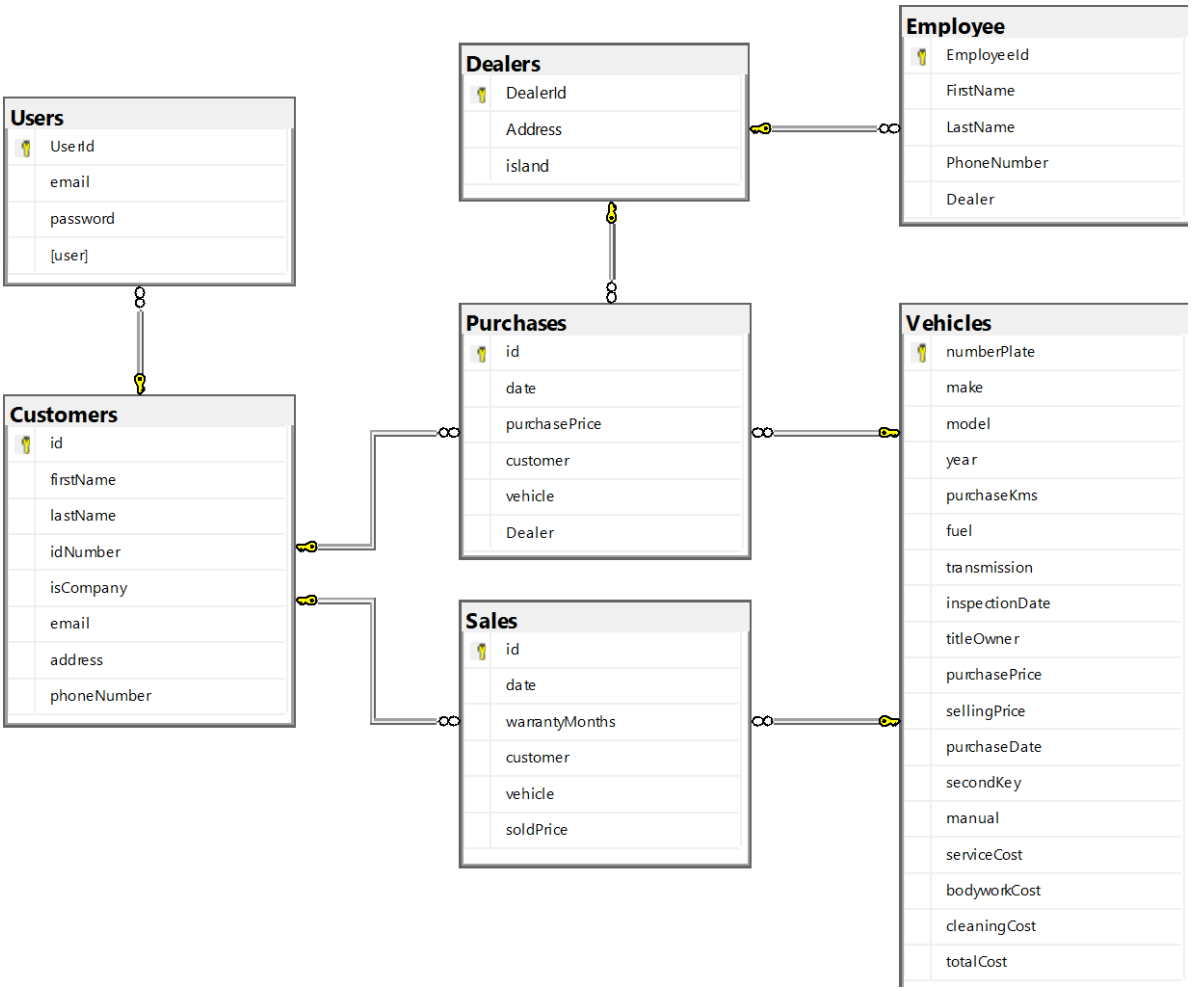


Ilustración 17 - Esquema de datos

3.5 Diseño de tests

En el diseño de software, se define como “testing” al proceso de evaluar y verificar que un producto o aplicación hace lo que se supone que tiene que hacer, previniendo bugs y detectando problemas durante la fase de diseño, hecho que implica que el coste de solventar el problema sea mucho menor que si se descubriese en una fase más tardía del desarrollo.

Existen varios tipos de tests, cada uno con objetivos específicos, con diferencias en el alcance del que pretenden comprobar el correcto funcionamiento, o diferencias en los tipos de comprobaciones que realizan. Algunos ejemplos son los siguientes:

- Tests de aceptación: Verificación de que todo el sistema funciona según lo previsto.

- Tests de integración: Garantizar que los componentes o funciones del software funcionan juntos.

- Tests unitarios: Validan que cada unidad de software funciona como se espera. Una unidad es el componente más pequeño de una aplicación que se puede probar, por ejemplo, un método concreto de una clase.

- Tests funcionales: Comprobación de las funciones mediante la emulación de escenarios de negocio, basándose en los requisitos funcionales. Las pruebas de caja negra son una forma habitual de verificar las funciones.

- Tests de rendimiento: Comprobación del rendimiento del software bajo diferentes cargas de trabajo. Las pruebas de carga, por ejemplo, se utilizan para evaluar el rendimiento en condiciones de carga reales.

- Tests de regresión: Comprobar si las nuevas características rompen o degradan la funcionalidad.

- Tests de estrés: Prueba de la carga que puede soportar el sistema antes de fallar. Se considera un tipo de prueba no funcional.

- Tests de usabilidad: Comprobar si un cliente puede utilizar un sistema o una aplicación web para realizar una tarea.

Además de los diferentes tipos de tests, existen diferentes metodologías para aplicarlos, en nuestro caso hemos usado el enfoque conocido como Test Driven Development o TDD.

Como su propio nombre indica, el TDD es un enfoque de desarrollo guiado por las pruebas, es decir, para implementar software primero se escriben las pruebas que indicarían su correcto funcionamiento y luego el código necesario para que la prueba sea exitosa. Simplificándolo, el TDD se puede definir en tres reglas:

1 No se permite escribir nada de código de producción a menos que sea para hacer pasar una prueba unitaria que falla.

2 No se permite escribir más de una prueba unitaria que esté fallando y su tamaño tiene que ser el mínimo posible que implique el fallo del código

3 No se permite escribir más código de producción que el suficiente para pasar la única prueba unitaria que falla.

Esto tiene dos grandes ventajas evidentes, la primera es que todo el código producido está testeado y la segunda es que se ha generado el código justo y necesario para cumplir los requisitos, sin excedencias.

Sin embargo sus bondades no se limitan a eso, si se sigue el ciclo Red-Green-Refactor que describe una forma de realizar el TDD de forma iterativa, generando un test en rojo (es decir, que falle), generando el código para que se pase el test y luego refactorizando la solución recién generada, se consigue un diseño del sistema muy optimizado, resultando en código más limpio, de alta calidad y como subproducto, se genera una documentación de los requisitos del sistema, ya que los propios tests están escritos en un lenguaje casi natural y recogen todas las necesidades que el sistema debería cubrir. (Beck, 2002)

4 Implementación

4.1 RCARS.Interface

Ya se ha mencionado con anterioridad que se ha utilizado lo que Microsoft llama Minimal API para que actúe de interfaz entre la app móvil y la base de datos, en este apartado se explicará con más detalle en que consiste.

Si utilizamos la plantilla proporcionada por Visual Studio para crear una Minimal API, esta nos generará un proyecto con un archivo Program.cs, cuya función es configurar compilar y ejecutar la aplicación, todo esto con solo unas cuantas líneas de código.

De hecho, incluso contiene los endpoints que se expondrán para que otras aplicaciones accedan a la API.

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.MapGet("/", () => "Hello World!");

app.Run();
```

En este mismo archivo se pueden implementar las instancias de los servicios que necesitemos, por ejemplo, se puede crear un Singleton del servicio que accede a la base de datos para así tenerlo preparado para su uso desde el primer momento. Si juntamos todo esto, el archivo quedaría de la siguiente manera:

```
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddSingleton<IVehiclesRepository,
CustomersRepository>();
builder.Services.AddScoped<ICustomersRepository,
CartRepository>();

var app = builder.Build();
app.MapGet("/vehicles", () => {
    ...
});
app.MapPost("/vehicles", () => {
    ...
});
app.MapGet("/vehicles/{PlateNumber}", () => {
    ...
});
app.MapGet("/customers",  () => {
    ...
});

app.Run();
```

Y casi sin darnos cuenta, ya tendríamos una API funcionando, con funcionalidades mínimas, pero funcionando al fin y al cabo.

Sobre esta base, mediante TDD y aplicando una arquitectura un poco más limpia para evitar que el código sea un monolito, se ha implementado la interfaz RCARS.Interface y ahora ofrece rutas a internet URIs (Endpoints) a los que cualquiera puede consultar mediante el protocolo HTTP, mas concretamente en la interfaz REST.

Simplificándolo mucho, REST es un conjunto de normas de comunicación entre dos aplicaciones. Si se quiere consultar todos los vehículos de la base de datos, simplemente basta con realizar una petición de tipo GET a la API con la ruta */vehicles*. En el caso de querer obtener un vehículo concreto la petición seria la misma, pero en la ruta habría que indicar el identificador del vehículo que queremos obtener, por ejemplo, si la matricula se usase como identificador la ruta sería */vehicles/8411KXK*.

Para URIs más complejas se usan varios niveles de anidación, si queremos obtener todos los vehículos de un cliente concreto, podríamos realizar la petición a la ruta */customers/{identificador del cliente}/vehicles*

Para insertar un vehículo en la base de datos se utilizan otro tipo de peticiones, POST. Y dado que en la ruta no se puede incorporar toda la información necesaria de un objeto complejo, se utiliza el cuerpo del mensaje de las peticiones HTTP para, en formato JSON, describir todas las propiedades del vehículo.

Otros tipos de peticiones son PUT para modificar y reemplazar, PATCH para modificar y crear una nueva versión y DELETE para eliminar un objeto.

Todas estas peticiones envían una respuesta que contiene un código que resume el resultado de la operación (p.e. 200 = OK, 404 = Not Found) y un cuerpo en el que mostrar un mensaje descriptivo de la operación o la respuesta esperada si fuera el caso, por ejemplo, si solicitamos un vehículo, el cuerpo de la respuesta en nuestro caso tendría este aspecto:


```
[
  {
    "numberPlate":"4567KDF",
    "make":"Toyota",
    "model":"Yaris",
    "year":2019,
    "purchaseKms":18500,
    "fuel":"HIB",
    "transmission":"A",
    "inspectionDate":"2022-05-21T00:00:00",
    "titleOwner":"CL",
    "sellingPrice":13990.0000,
    "purchasePrice":11500.0000,
    "purchaseDate":"2022-05-20T00:00:00",
    "secondKey":true,
    "manual":true,
    "serviceCost":130.0000,
    "bodyworkCost":65.0000,
    "cleaningCost":50.0000,
    "totalCost":11745.0000
  }
]
```


Por parte de los test, realizados sobre una base de datos de estructura idéntica a la base de datos funcional, cabe destacar el uso del paquete FluentAssertions, que como su propio nombre indica, hace que los “asserts” de los tests, es decir, las comprobaciones, sean escritas en un lenguaje de muy alto nivel, cercano incluso al natural. La muestra se puede ver en el siguiente ejemplo de test.

```
[Test]
public async Task ReturnAllVehicles()
{
     await using var application = new WebApplicationFactory<Program>();
     using var client = application.CreateClient();

     var response = await client.GetStringAsync("/Vehicles");

     var vehicles = response.Split("},{");
     vehicles.Should().HaveCount(2);
     vehicles[0].Should().Contain("\"numberPlate\":\"4567KCF\"");
     vehicles[1].Should().Contain("\"numberPlate\":\"8511KXK\"");
}
```

Por claridad, los tests se separan en 3 partes, dejando una línea vacía en medio de estas. Las tres partes se conocen como la triple A, Arrange (Preparación del test), Act (La acción que se quiere testear y los anteriormente mencionados Assert (Afirmar que los resultados fueron correctos).

Si vemos las 3 últimas líneas del Assert, incluso una persona que no sepa de código podría intuir lo que está comprobando el test. Además, por conveniencia y convención, las clases de test se denominan como la clase que se quiere probar seguido del verbo “Should”, mientras que los tests individuales se nombran con el comportamiento concreto a testear.

Esto hace que la realización de pruebas sea mucho más entendible y legible, ya que, si tomamos como ejemplo el test de la figura anterior, sumado al nombre de su clase, lo que veríamos es: VehiclesControllerShould.ReturnAllVehicles() y viendo esto, ya sea en un explorador de pruebas o en consola, ya sabríamos exactamente que esta fallando en nuestro programa.

4.2 RCARS Companion

La app móvil, llamada RCARS Companion, posee la arquitectura típica de las apps Java o Kotlin diseñadas específicamente para dispositivos móviles. Por un lado tenemos Activities o Fragments que son elementos encargados de la interfaz con el usuario y están asociados a vistas, normalmente llamadas de forma similar a la Activity, y por otro lado tenemos la lógica de negocio y los modelos, cuyas clases pueden ser accedidas desde las actividades, pero para mantener una limpieza de código han de estar separadas de estas.

En la siguiente pieza de código se puede ver la simpleza con la que se le asigna la función a un botón de pasar de una actividad a otra, en este caso del menú principal a la pantalla de Mapas, para mostrar la ubicación de un taller.

```
btnWorkshops = findViewById(R.id.itemWorkshop);

...

btnWorkshops.setOnClickListener(v -> {
    Intent intent = new Intent(MainActivity.this, MapsActivity.class);
    intent.putExtra("Show","Workshops");
    startActivity(intent);
});
```

En cuanto a funcionalidad, los stakeholders dejaron claro desde un principio la necesidad de que la aplicación posea una interfaz intuitiva, clara y fácil de usar. Es por esto por lo que se optó por un menú principal con todas las "*features*" visibles nada más realizar el inicio de sesión.

Estas *features* son ver los coches disponibles, acceder a información de mi/s vehículo/s, contactar con el concesionario, obtener información de contacto de un taller, etc. Y prácticamente todas ellas necesitan datos alojados en el servidor, por tanto, la forma de acceder a estos datos es muy importante.

Ya se ha hablado de como la API accede a estos datos y expone URIs para ser accedida, sin embargo, es igual de interesante la forma en la que la app Android accede a estas URIs.

No resultaría cómodo tener que montar una petición HTTP con una ruta y cuerpo cada vez que sea necesario, por ejemplo, obtener datos del cliente. Ahí es donde entra Retrofit. Es un cliente de servidores REST desarrollado por Square y permite hacer todo tipo de peticiones, así como gestionar los diferentes parámetros y conversión de la respuesta a un modelo del ámbito de la aplicación. Y para ello hacen falta solo dos clases: un servicio que convierta las peticiones en simples métodos que poder llamar fácilmente y que se encargue de la transformación de la respuesta y otra clase que sirva para instanciar la interfaz del servicio con una conexión concreta, una clase que actúa como un adaptador.

Para verlo más claro aquí se muestran la clase que instancia la interfaz y la interfaz propiamente dicha, esta última de forma simplificada.

Un método de la interfaz que accede a una URI:

```
@GET("login")
Call<LoginResponse> getLogin(
        @Query("username") String username,
        @Query("password") String password
);
```

La instancia concreta de la interfaz con la conexión a la API, como podemos ver, alojada en Azure:

```
public class APIAdapter {
    private static APIService API_SERVICE;

    public static APIService getApiService() {

        // Creamos un interceptor y le indicamos el log level a usar
        HttpLoggingInterceptor logging = new HttpLoggingInterceptor();
        logging.setLevel(HttpLoggingInterceptor.Level.BODY);

        // Asociamos el interceptor a las peticiones
        OkHttpClient.Builder httpClient = new OkHttpClient.Builder();
        httpClient.addInterceptor(logging);

        String baseUrl = "https://rcarsapi.azure-api.net/";

        if (API_SERVICE == null) {
            Retrofit retrofit = new Retrofit.Builder()
                    .baseUrl(baseUrl)
                    .addConverterFactory(GsonConverterFactory.create())
                    .client(httpClient.build()) // <-- usamos el log level
                    .build();
            API_SERVICE = retrofit.create(APIService.class);
        }

        return API_SERVICE;
    }
}
```

Con estas dos clases ya podemos acceder a la API de una forma cómoda y sencilla, bastaría con invocar los métodos definidos en la interfaz de forma asíncrona.

5 Resultado

5.1 RCARS Companion

Para mostrar el resultado conseguido en cuanto a la aplicación móvil se refiere, lo más conveniente es enseñar el funcionamiento de la misma, en este caso se mostrarán capturas de pantalla explicando las diferentes tareas que se están ejecutando y la relación que tienen con las historias de usuario de las que provienen.

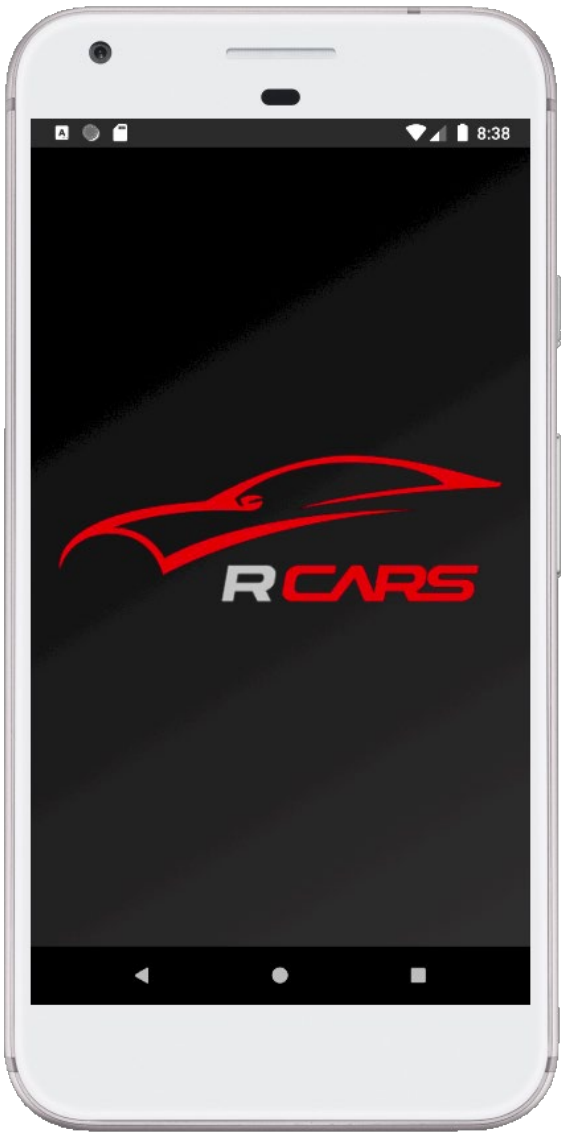
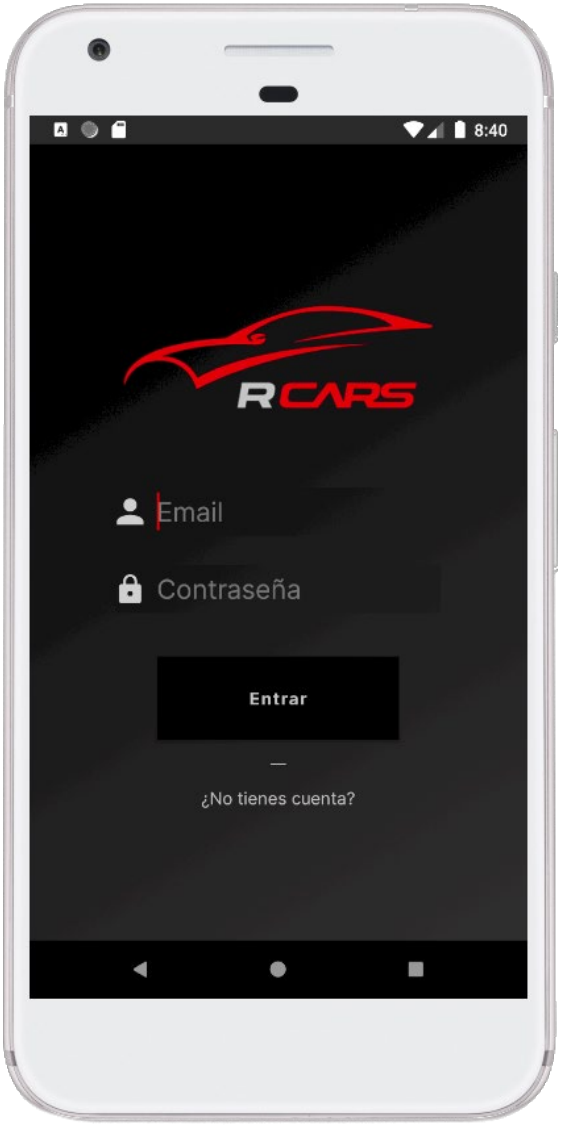


Ilustración 18 - Captura RCARS Companion I

En estas capturas podemos ver la pantalla de carga y la de inicio se sesión.

La visualización del degradado como imagen de fondo se ve escalonada por limitaciones del emulador.



La transición entre una pantalla y otra se realiza mediante una animación, ya que para el product owner era muy importante que el producto tenga una calidad percibida alta desde el principio.

Ilustración 19 - Captura RCARS Companion II

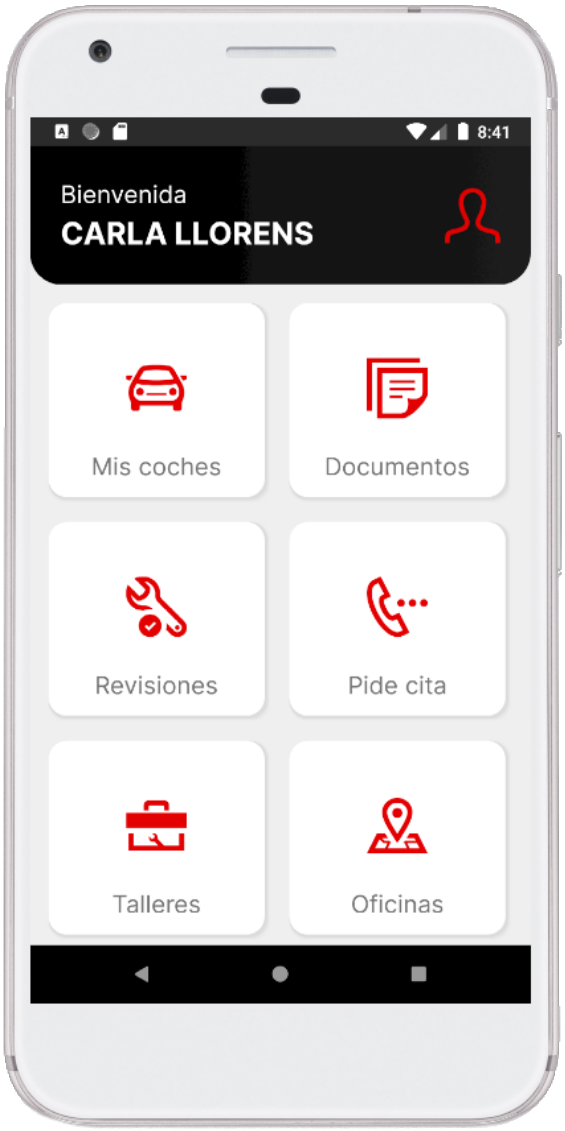


Ilustración 20 - Captura RCARS Companion III

Una vez se ha iniciado la sesión se accede al menú principal donde nos recibe un mensaje de bienvenida personalizado.

En este menú principal, podemos ver y acceder a todas las funciones principales de la aplicación.

Ilustración 21 - Captura RCARS Companion IV

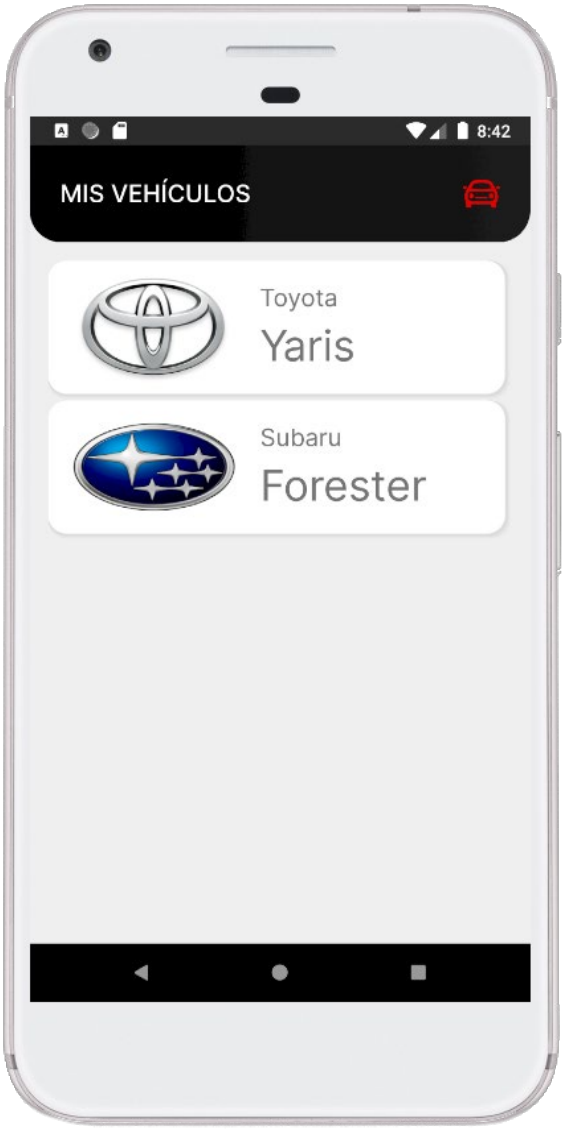


Ilustración 22 - Captura RCARS Companion V

En la pantalla “Mis Vehículos”, que se accede desde el botón superior izquierdo del menú principal, podemos ver todos los vehículos que el usuario ha adquirido con anterioridad

Si toca alguno de los vehículos podrá ver más detalles del mismo. Algo tan sencillo parece no ser de mucha utilidad, pero según los empleados, esta información ahorraría alrededor de un tercio de las llamadas que reciben de clientes que han adquirido un vehículo.

Ilustración 23 - Captura RCARS Companion VI

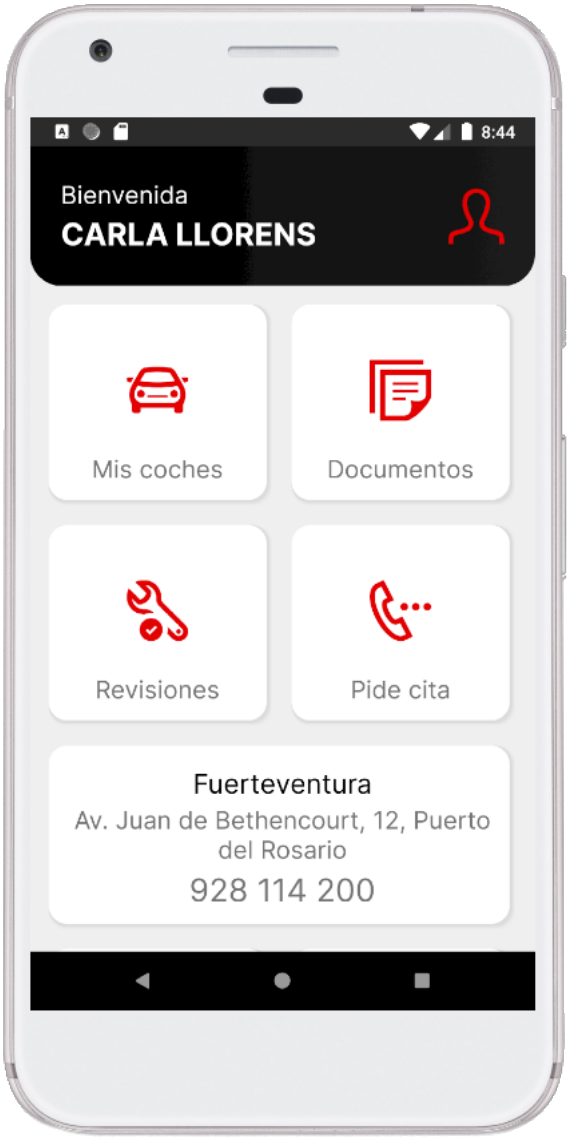
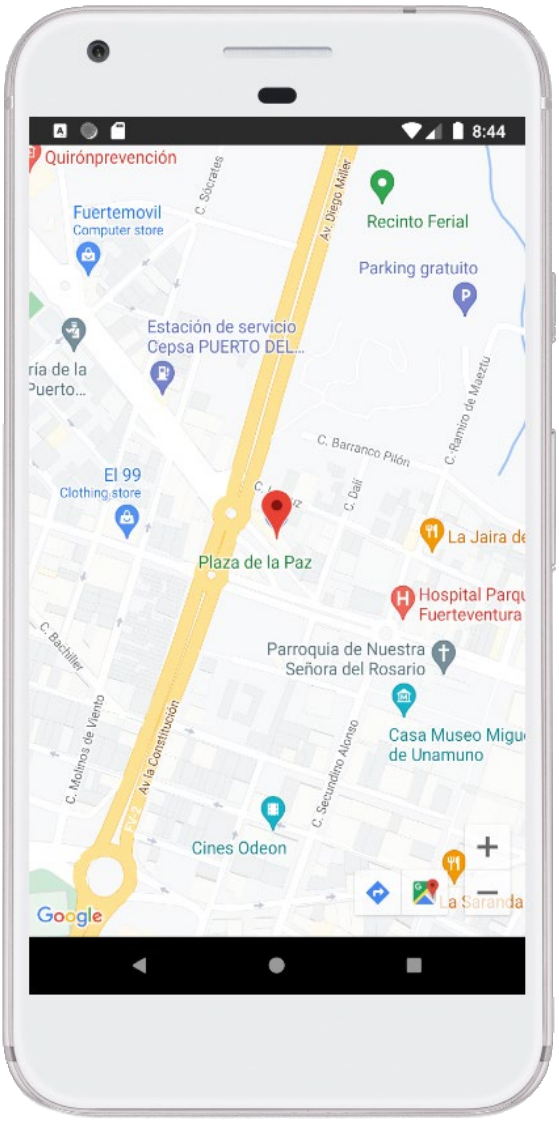


Ilustración 24 - Captura RCARS Companion VII

Aquí se puede observar la acción “Pide Cita”, que abre un submenú con información de los diferentes concesionarios y sus teléfonos y direcciones, obteniendo los datos de la base de datos.

Así mismo, como se ve en la siguiente ilustración, desde las secciones del menú “Oficinas” y “Talleres” se dirige al usuario a la ubicación del taller o de la oficina, ofreciendo si lo desea indicaciones para llegar desde su ubicación.



Estas dos funcionalidades son claros ejemplos de fases previas de desarrollo, donde en iteraciones futuras se ampliará la funcionalidad, en la captura anterior mostrando la posibilidad de pedir cita directamente desde la app y en esta, mostrando primero un listado de los diferentes concesionarios y talleres antes de ser dirigidos hacia ellos.

Ilustración 25 - Captura RCARS Companion VIII

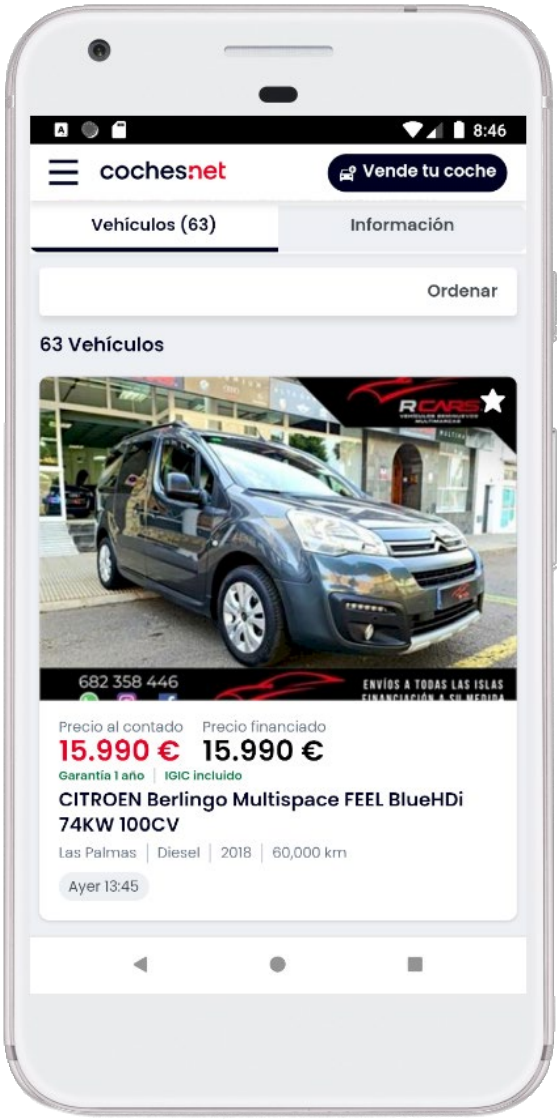
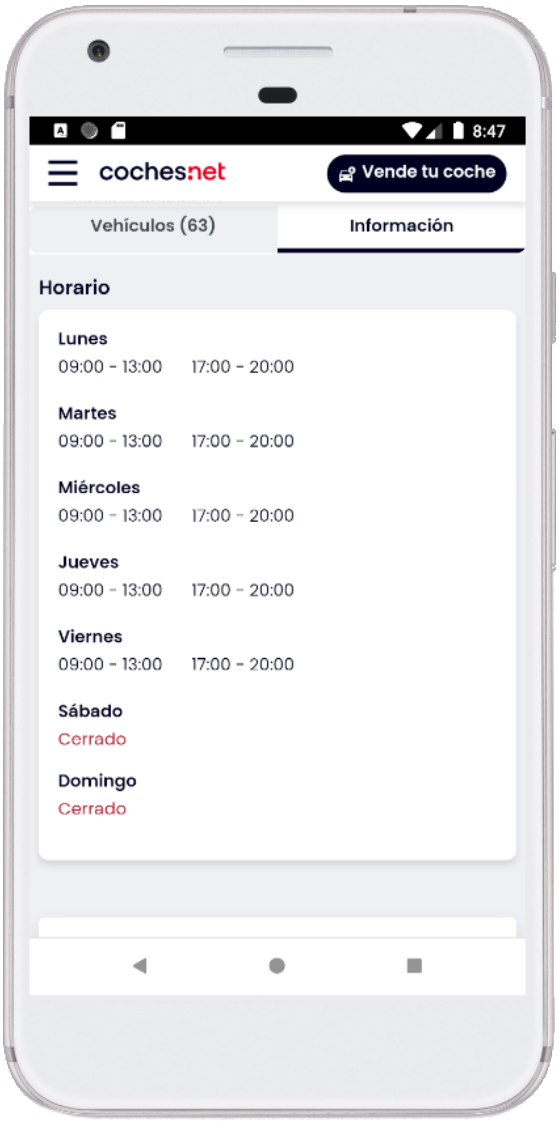


Ilustración 26 - Captura RCARS Companion IX

El último "botón" del menú principal es uno de los más importantes, ya que lleva a la web donde ver todos los coches que hay en stock en el concesionario y la información de los mismos.



También puede acceder a información general del concesionario como el horario

Ilustración 27 - Captura RCARS Companion X

5.2 RCARS Interface

En cuanto la API, el resultado es una herramienta que cumple con los patrones arquitectónicos de REST y se situaría en el segundo nivel de madurez en la escala de Richardson, ya que ofrece una URL para los recursos de forma individual y hace uso de los verbos HTTP. De momento no es necesario llevarla al siguiente nivel con controles de hipertexto, a veces referido como HATEOAS (debido a sus siglas en inglés Hypertext As The Engine Of Application State) ya que este nivel responde a las peticiones con las URLs de los posibles siguientes pasos, para que los flujos de acciones sean más rápidos, sobre todo en APIs de negocio que son accedidas por otros servicios. Sin embargo, este no es nuestro caso ya que serían los usuarios los que realicen las peticiones. (Fowler, 2010)

Aquí podemos ver el Swagger de la API con todos sus endpoints actuales

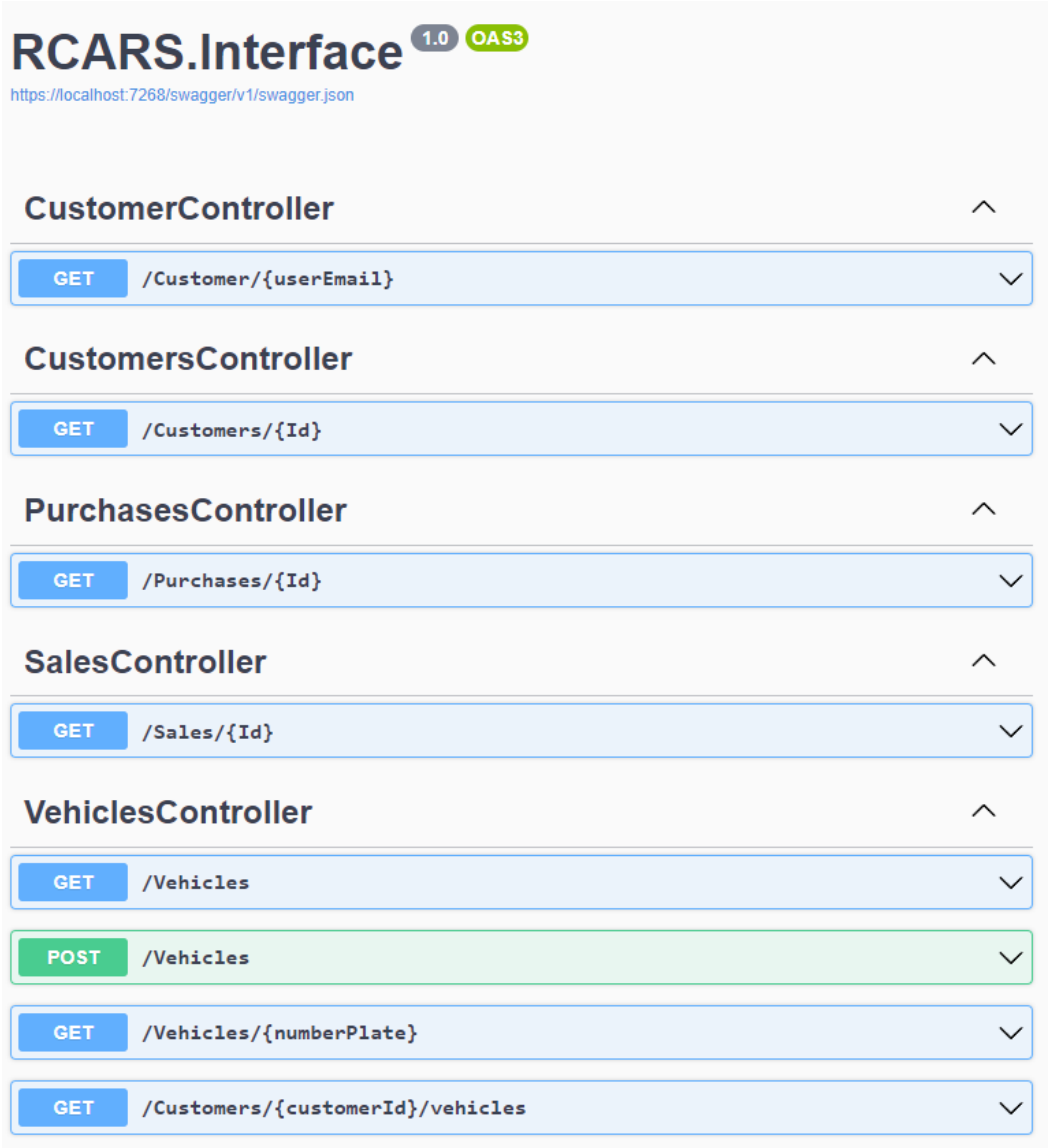


Ilustración 28 - RCARS.Interface/Swagger

Además, al implementarse mediante TDD, podemos tener la certeza y tranquilidad que cualquier cambio futuro no romperá la funcionalidad ya implementada puesto que hay pruebas para comprobarla.

6 Trabajos futuros y perspectiva

Desde un principio se ha planteado este proyecto con la idea de no finalizarse en las 300 horas que requiere. La intención era poder sentar las bases y se ha conseguido, pero el alcance deseado es bastante mayor al alcanzado en el estadio actual.

Lo más evidente es la finalización de la aplicación móvil RCARS Companion, la gestión de documentos, poder pedir cita a través de la app, ver tarjetas de presentación de los trabajadores de cada tienda y otras funcionalidades se han quedado fuera de este trabajo, pero no fuera del plan establecido y se verán implementadas en futuras *releases,* así como la mejora de las prestaciones que ya están incluidas.

Por otro lado, fue seleccionada una estructura basada en servicios con el fin de que sea accedida por más de una aplicación. RCARS Interface servirá para nutrir de datos a un software de escritorio, destinado a los trabajadores desde donde se cargará y consultará la información y documentos de las compras y ventas, en detrimento del flujo actual en el que es necesario acceder directamente a la base de datos para que se vea reflejado en la app, solución provisional a la que se acudió con el fin de entregar un MVP de la aplicación móvil en el menor tiempo posible.

Esta futura aplicación, además de ser un portal de datos de las transacciones que realizan los trabajadores, servirá para contabilizar los diferentes stocks de cada concesionario, así como las estadísticas de venta de cada trabajador. Estas soluciones, y otros aspectos como el que en la base de datos exista una entidad “Dealer”, se deben a que está prevista la apertura de dos concesionarios más pertenecientes a la marca en el primer trimestre de 2023.

Otro aspecto que mencionar es la intención de colaborar con diferentes empresas locales. Estableciendo relaciones con comercios integrados en el sector de la automoción o interesados en llegar a este nicho de mercado. Ambas empresas podrían beneficiarse, por ejemplo, vendiendo espacio publicitario dentro de RCARS Companion a empresas de lavado de vehículos, comercios locales, talleres, etc.

El objetivo final es ofrecer una suite de aplicaciones con las que se facilite el trabajo a los empleados, se ofrezca información a los encargados de tomar decisiones y se preste un mejor servicio a los clientes, todo esto con la modularidad adecuada para poder escalar en mayor o menor medida las diferentes partes del sistema ajustándose a las necesidades reales de la empresa.

Hablando de la parte más técnica, también está pensado desarrollar una automatización del despliegue y de la integración de las aplicaciones, lo que se conoce como CI/CD por las siglas de Continous Integration and Continous Delivery. Mediante las herramientas

que ofrece Azure y aprovechando que ya el producto está integrado en la plataforma de Microsoft, resulta relativamente sencillo automatizar la producción de paquetes y el despliegue de los mismos mediante pipelines y releases.

Con esto se consigue que, por ejemplo, tras hacer un push a la rama master del repositorio, se lance una comprobación de los tests del programa, se ejecuten herramientas de evaluación de código como Sonarqube, se genere el paquete y se guarde en un contenedor de Azure y tras ello se despliegue la aplicación en un entorno de preproducción o QA. Todo de forma automática, haciendo que el valor percibido (mediante las nuevas funcionalidades) llegue antes al cliente y de mejor forma (libre de errores, con código limpio, etc.)

7 Conclusiones

Para concluir la memoria se evaluará lo que se ha conseguido con el trabajo, desde el producto que se ha desarrollado hasta las habilidades y conocimientos que se han adquirido.

Por una parte, a pesar de que no haya dado tiempo a finalizarla, se ha logrado llegar a la fase de desarrollo esperada en un principio a la aplicación móvil. Esta ofrece las funcionalidades que esperaban los stakeholders, como ofrecer información de los vehículos adquiridos, de los disponibles y de las tiendas, y ya las versiones más tempranas poseen valor puesto que ahorrarían trabajo a los vendedores y ayudarían a los clientes, ahorrándoles llamadas a la tienda.

Por otro lado, RCARS Interface ofrece, como su propio nombre indica, una interfaz para trabajar con la base de datos y debido a su estructura resulta sencillo escalarla y añadir funcionalidades en cuanto sean necesarias ya sean para RCARS Companion u otra futura aplicación diferente.

En cuanto a conocimientos, el hecho de trabajar con tantas tecnologías y lenguajes de programación diferentes ha resultado muy enriquecedor. C#, .Net, Java, consultas SQL y REST, Retrofit, Entity Framework, Azure, Postman… de algunas de estas herramientas se tenían conocimientos previos, pero sin duda resulta muy evidente la mejoría en el manejo de estas, así como de técnicas como el TDD o la refactorización.

Sin embargo, donde más conocimientos se han adquirido es sin duda durante las fases previas del desarrollo, tanto la obtención de requisitos, como la planificación de las releases y la inmersión en el negocio con los stakeholders en general, han resultado ser tremendamente didácticas y han permitido trabajar y adquirir conocimientos muchas veces olvidados en el ámbito del desarrollo software, pero no por ello menos importantes, ya que con una comunicación pobre con los clientes, un proyecto está destinado al fracaso.

8 Bibliografía y referencias

Beck, K. (2002). *Test Driven Development: By Example.*

Fowler, M. (2010). *Richardson Maturity Model*. Obtenido de https://martinfowler.com/articles/richardsonMaturityModel.html#level3

Ganvan. (2021). *Boletines estadísticos Ganvam.*

Hürsch, W. L., & Lopes, C. V. (1995). *Separation of concerns.*

Maurya, A. (2014). *Running Lean: cómo iterar un plan A a un plan que funciona.* UNIR.

Patton, J. (2015). *Story Map Concepts.* Obtenido de https://www.jpattonassociates.com/wp-content/uploads/2015/03/story_mapping.pdf

Pérez, J. S. (s.f.). *Fundamentos de la Arquitectura del Software.* Obtenido de https://ncvt-aep.ulpgc.es/cv/ulpgctp21/pluginfile.php/401938/mod_resource/content/6/Tema%201%20-%20Fundamentos%20de%20AS.pdf

ULPGC. (s.f.). *Arquitectura orientada a servicios.* Obtenido de https://ncvt-aep.ulpgc.es/cv/ulpgctp21/pluginfile.php/401331/mod_resource/content/22/Arquitectura%20orientada%20a%20servicios%20v2020.pdf

ULPGC. (s.f.). *MODELADO CONCEPTUAL DEL DOMINIO.* Obtenido de https://ncvt-aep.ulpgc.es/cv/ulpgctp21/pluginfile.php/403144/mod_resource/content/1/2019%2004%20REVISI%C3%93N%20MODELADO%20CONCEPTUAL.pdf

ULPGC. (s.f.). *PLANIFICACIÓN USANDO HISTORIAS DE USUARIO.* Obtenido de https://ncvt-aep.ulpgc.es/cv/ulpgctp21/pluginfile.php/403155/mod_resource/content/1/2019%2012%20PLANIFICACI%C3%93N%20USANDO%20HISTORIAS%20DE%20USUARIO.pdf

Andrew H. & Thomas D. (1999). *The Pragmatic Programmer, From Journeyman to Master*

9 Índice de ilustraciones