Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN TECNOLOXÍAS DA INFORMACIÓN

Aplicación para la gestión de los vehículos y sus reparaciones en un taller

Estudante: Noelia Cotelo Garrido
Dirección: Laura M. Castro Souto

A Coruña, xuño de 2020.

Nunca te rindas porque el esfuerzo valdrá la pena

Agradecimientos

No podría haber llegado a donde estoy si no fuese por todas las personas que me rodean, las cuales de una forma u otra me han ayudado en este camino. Por este motivo, quiero agradecer a mi familia y a mis amigos todo el apoyo que me han dado estos años.

Especial mención a mis padres, ellos han sufrido conmigo en los malos momentos, ayudándome a levantarme siempre que me he caído. También han sido los primeros en alegrarse con cada asignatura aprobada, cada práctica conseguida y cada buena noticia que recibía.

A mi hermano, por haberme dado la idea de este proyecto y estar ahí siempre que lo he necesitado.

A Álex y Lau porque sin su apoyo incondicional, nada de esto hubiese sido posible.

Por último, agradecer a Laura por guiarme en este proyecto y prestarme su ayuda siempre que lo he necesitado.

Resumen

El objetivo de este proyecto consiste en el desarrollo de una aplicación móvil multiplataforma para ayudar a la gestión de la atención al público de un taller mecánico dando soporte a la asignación de citas previas y a la recepción de facturas. La aplicación permitirá la creación de usuarios y sus vehículos, la concertación de revisiones periódicas o reparaciones puntuales, así como la generación de facturas para que los usuarios las puedan consultar desde sus dispositivos en el momento en el que estén disponibles.

El sistema está basado en una API REST que da acceso a la lógica de negocio, y se ha implementa usando tecnologías J2EE y apoyándose en el *framework* Spring MVC. Se utiliza PostgresSQL como base de datos relacional para persisitir los datos de la aplicación. Para la parte del *frontend* se ha elegido Ionic framework, que permite a la aplicación móvil ser multiplataforma, esto es, poderse ejecutar indiferentemente en Android, iOS o Windows Phone.

Abstract

The goal of this project is the development of a multi-platform mobile application to help manage the customer service of a garage by supporting the assignment of appointments and the management of bills. The application will allow the creation of users and their vehicles, the arrangement of periodic checks or specific repairs, as well as the generation of bills that users can consult from their devices as soon as they are available.

The system is based on a REST API that gives access to business logic, and has been implemented using J2EE technologies and supported by the Spring MVC framework. PostgresSQL is used as a relational database to persist the data of the application. For the frontend part, the Ionic framework has been selected, which allows the mobile application to be multiplatform, and run independently on Android, iOS or Windows Phone.

Palabras clave:

- Taller mecánico
- Reparación de vehículos
- Aplicación móvil
- Java
- Spring MVC
- PostgresSQL
- Ionic
- Scrum
- API REST

Keywords:

- Garage
- Car repair
- Mobile application
- Java
- Spring MVC
- PostgresSQL
- Ionic
- Scrum
- API REST

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Actualmente nuestro mundo gira en torno a internet, cada día que pasa se vuelve más difícil querer un producto y no tenerlo en nuestro hogar al cabo de unos días gracias a nuestro *smartphone*. Nuestro coche no podía quedarse al margen de toda esta inmediatez y en este contexto surge el presente trabajo fin de grado, con el objetivo de crear una aplicación que facilitase el tedioso proceso de llevarlo al taller y realizar las gestiones necesarias relacionadas con la visita y el diagnóstico.

1.1 Objetivos

El objetivo de este proyecto es crear una aplicación web adaptada para su uso en dispositivos móviles que facilite la gestión de un taller de vehículos al tiempo que ofrezca a los dueños de los vehículos una herramienta que le permita ver los detalles de su coche, concertar citas y visualizar las facturas. Además, los usuarios registrados tendrán acceso a promociones gracias a las cuales obtendrán beneficios en sus facturas.

Para la parte de gestión del taller existirán usuarios administradores que, además de gestionar a los usuarios registrados, podrán hacer actualizaciones sobre el estado de las citas agilizando así las comunicaciones con el cliente.

La aplicación va a estar orientada a todo tipo de usuarios que posean un coche. Al no haber un perfil concreto, el objetivo en cuanto a interfaz es el de crear un entorno fácil de usar e intuitivo para que el usuario lo pueda usar cómodamente.

1.2 Evaluación de alternativas existentes

En esta sección se analizan algunas de las aplicaciones existentes en el mercado y cuyas funcionalidades son similares a las implementadas en el proyecto actual.

A priori se podría creer que no existen herramientas que realicen las tareas de gestión de un establecimiento de este tipo ya que no es un mercado que se encuentre en el foco de la actualidad. Sin embargo, y a continuación queda demostrado, existen varios sistemas que agilizan las gestiones referentes a los vehículos de manera similar a la que se propone en este proyecto.

1.2.1 iTaller

Esta aplicación lanzada por la empresa Connection Soft Service [1] ayuda a controlar tareas que van desde la recepción del vehículo hasta el control de los trabajos en curso. Las principales funciones de esta aplicación por parte de recepción y gerencia son el alta de vehículos y clientes así como la disponibilidad de una agenda de cita previa. Además, permite realizar firma electrónica aportando seguridad y agilidad a los procesos de firma y creación de documentos en PDF. Para la parte de la gestión, como añadido, permite realizar un control sobre los trabajos en curso y visualizar un cuadro de mando para evaluar los datos de la empresa. Por otra parte, la empresa que desarrolla la aplicación, ofrece la posibilidad de comunicar la aplicación con otros programas de gestión de talleres para facilitar el intercambio de datos entre los dos sistemas, así como la adaptación de los procesos existentes en la aplicación a las necesidades propias del negocio.

Figura 1.1: iTaller para el usuario

1.2.2 Autingo

Esta aplicación [2] se basa en facilitar un presupuesto a los conductores sin la necesidad de acudir al establecimiento. Una vez obtenido el presupuesto, el usuario puede reservar una cita en el taller que desee. Esta aplicación permite registrarse al usuario y a su vehículo y a continuación da la opción de seleccionar el tipo de servicio que se va a realizar entre reparación o mantenimiento para acotar la estimación de precio que va a recibir. Además, a través

Figura 1.2: iTaller para el administrador

de la geolocalización se muestran los talleres que hay a su alrededor para poder elegir la cita que mejor le convenga al cliente (ver figura 1.3).

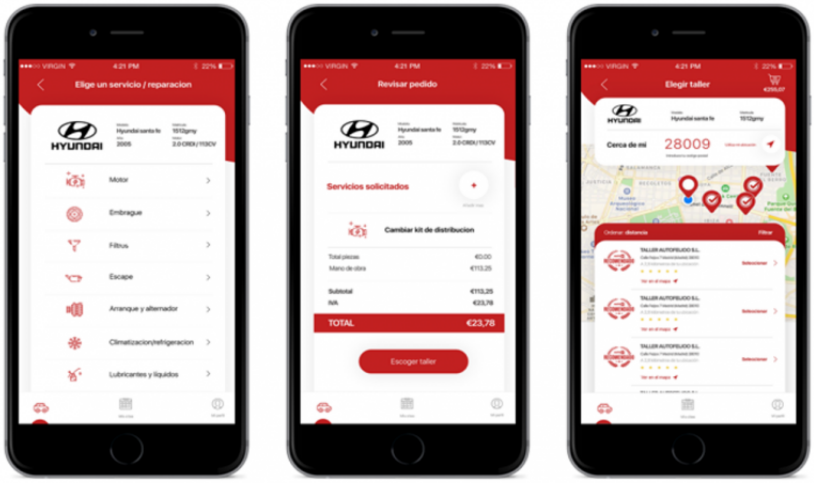


Figura 1.3: Autingo

1.2.3 Comparativa

La aplicación *Autingo* ofrece la posibilidad de reservar una cita previa cuando el usuario lo desee además de elegir el establecimiento al que acudir en base a la posición actual del cliente. Sin embrago, los usuarios critican la falta de comunicación con la plataforma de Autingo además de tener que desembolsar más dinero del que se había pactado al inicio por la aparición

de problemas durante el proceso de la reparación.

A raíz de estas opiniones sobre la aplicación, se ha deducido que los usuarios pueden no identificar de forma correcta los síntomas de la avería para comunicarlo a través de la aplicación. Este hecho provoca que se diagnostique un tipo de avería y cuando el especialista revisa el vehículo, se encuentra con que es otro problema totalmente diferente, generando pérdidas al establecimiento ya que la avería será de precio más elevado que el presupuestado inicialmente. Por eso, en nuestra plataforma apostamos por una cita de media hora donde un profesional cualificado evalúe los posibles daños del vehículo y así, categorice la reparación de manera correcta. Además, aunque la información está disponible en ambas plataformas las 24 horas del día, los usuarios de Autingo se quejan de que les han llegado a cancelar la cita sin previo aviso [3]. Esto ha llevado a desarrollar un sistema de notificaciones para que, cualquier cambio en el estado de su cita sea notificado al usuario para que esté al corriente. Por último, el servicio de iTaller es de pago a través de la empresa CSS. Los pequeños negocios no pueden costearse ese gasto por lo que un servicio así les abriría mucho mercado ya que, en los tiempos que corren, esperar no es una opción. La aplicación de este proyecto tendrá licencia libre para facilitar el acceso a pequeñas y medianas empresas y mejorar así sus negocios.

1.3 Metodología

En este proyecto, se ha decidido apostar por una de las metodologías ágiles más usadas en el mundo, *Scrum* [4]. Esta metodología se presentó en 1995 y desde ese momento, la mayoría de equipos de desarrollo lo usan. La razón de esta elección radica en los buenos resultados que da en todos las empresas en las que se implementa, elevando su productividad hasta en 10 veces más [5]. Además, es una metodología fácil de entender, lo cual no implica que sea fácil de implementar.

A continuación se explican las características de esta metodología y su adaptación al proyecto actual.

1.3.1 Scrum

Scrum [6] es un modelo que permite desarrollar software de manera ágil. En el proyecto actual, el equipo de desarrollo está formado por una sola persona, la propia alumna, por lo que ha sido necesario adaptar la metodología a las circunstancias.

Roles

El modelo de equipo Scrum se centra en construir software de calidad para lo cual existen tres roles:

- **Scrum Master**. Las funciones de esta persona dentro del equipo son dos, gestionar el correcto funcionamiento de Scrum y elminiar los impedimentos que puedan causar retrasos a la hora de entregar el producto.

- **Product Owner**. Persona que reparte el trabajo definiendo las prioridades y aportando valor al trabajo que se está realizando.

- **Equipo**. Grupo de profesionales con los conocimientos necesarios para realizar el desarrollo del proyecto.

Como se mencionaba anteriormente, en el proyecto actual, la alumna ha llevado a cabo funciones de *Scrum Master* y *Producto Owner* a parte del propio desarrollo. Esto ha ayudado a tener una visión más amplia de las tareas necesarias para realizar un proyecto y todas las decisiones y gestiones en las que un desarrollador no suele tomar parte. Además, la directora del proyecto también ha participado en los roles de *Scrum Master* y *Producto Owner* aportando ideas sobre las funciones a implementar en la aplicación y ayudando a resolver los problemas encontrados durante el desarrollo.

Eventos

Uno de los factores determinantes del buen funcionamiento de la metodología son los eventos ya que en ellos radica uno de los grandes pilares del éxito de los proyectos, la comunicación. Los eventos que conforman Scrum son los siguientes:

- **Sprint**. Es el evento principal sobre el que se construye el framework. Su duración suele oscilar entre 2 semanas y un mes.

- **Sprint Planning**. Se trata de una reunión programada al inicio de cada *sprint* en la que el Product Owner, el Scrum Master y el equipo de desarrollo definen qué se va entregar al final de la iteración que corresponda y cómo se va a estructurar el trabajo para poder finalizarlo con éxito. Además, se reordena el *Product Backlog* en base a las necesidades que el cliente haya manifestado.

- **Daily Scrum**. Esta reunión se realiza de forma diaria con una duración de unos 15 minutos. En ella, se lleva a cabo un ejercicio de transparencia para comprobar el avance diario del proyecto permitiendo una puesta en común de posibles problemas que estén dificultando el avance programado. De esta manera, si un compañero conoce la solución del problema de otro, se ahorraría mucho tiempo de análisis. Debido a que el equipo de desarrollo de la aplicación actual se conforma de una única persona, se ha prescindido de esta reunión.

- **Sprint Review**. Este tipo de reunión se realiza al final de cada sprint para revisar el rendimiento generado y pueden participar personas externas al equipo que el *Producto Owner* haya considerado oportuno incluir. Se suele mostrar el producto en funcionamiento para resolver las dudas sobre su uso además de explicar qué puntos de la planificación se han cumplido y cuáles no.

- **Sprint Restrospective**. Todo el equipo Scrum participa en esta reunión que se utiliza para analizar la aplicación de la metodología durante cada Sprint. La finalidad de este evento es identificar los puntos fuertes y los puntos débiles en la relación del equipo, las herramientas que se usan y las tareas realizadas. De esta forma, se intentan reforzar las tareas realizadas de forma correcta e identificar los errores cometidos para no volver a repetirlos en futuros *sprints*.

Dentro del proyecto, se han realizado reuniones al finalizar cada *sprint* en las que ha participado la directora y la alumna. En ellas se ha revisado el trabajo realizado hasta el momento y se han marcado los objetivos de desarrollo para el siguiente *sprint*. También se han identificado ciertos aspectos a mejorar, los cuales, sin usar esta metodología de seguimiento no se hubiesen podido detectar. Al ser un equipo de trabajo reducido, se han adaptado las diferentes reuniones que se suelen tener en una única optimizando así el tiempo.

Artefactos

Los artefactos son todos los elementos que facilitan el registro de la información fundamental del proceso de Scrum y que garantizan transparencia en las tareas realizadas dentro del equipo. Se trata del *Product backlog*, el *Sprint Backlog* y los *incrementos*.

- El **Product Backlog** es una lista de requerimientos, casos de uso, tareas y dependencias. Es una lista que puede evolucionar durante el desarrollo a medida que aparezcan necesidades.

  - Las **historias de usuario** son la técnica de toma de requisitos más usada en las metodologías ágiles. En ellas se proporciona información sobre cómo se comporta el requerimiento en el que se está trabajando. Son tomadas como requerimientos oficiales y se suelen expresar desde el punto de vista del usuario.

- El **Sprint Backlog** es un conjunto de elementos del *Product Backlog* que van a ser realizados durante un *sprint*.

- Los **incrementos** son la forma en la que se mide el progreso que ha habido en cada etapa. Es esencial que cada iteración tenga un incremento ya que si esto no ocurre, significa que ha fallado algo durante la metodología.

Para la gestión de los artefactos se ha usado la herramienta *Trello* [7] que permite la creación de tableros con una lista de las tareas a realizar (ver figura 1.4). Se ha creado un tablero con las tareas de cada *sprint* incluyendo la descripción de cada uno de ellos y una *checklist* para comprobar el avance de cada tarea individual (ver figura 1.5).

Gracias a la distribución de la herramienta, se mueven las tareas en tres columnas: *Nuevas, En Progreso* y *Realizadas* lo que permite ver el progreso de las tareas a simple vista. Además existe una columna para colocar las tareas bloqueadas y abordarlas en las reuniones pertinentes.

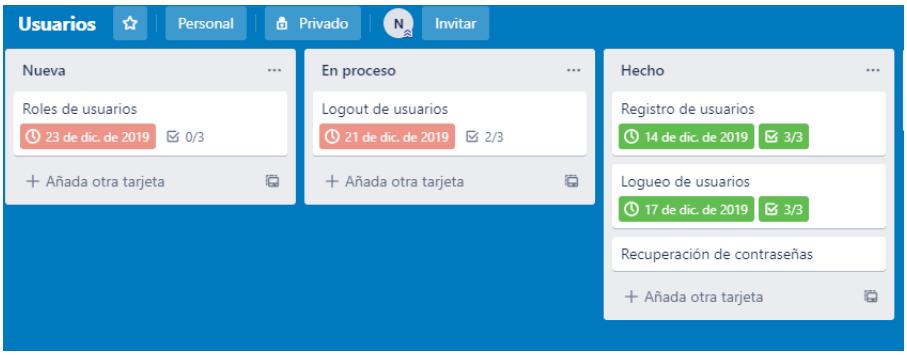


Figura 1.4: Tablero Trello

1.4 Fundamentos tecnológicos

En este punto, se va a dar una visión global de las tecnologías usadas en el desarrollo. Existen tres grupos de tecnologías según la forma en la que se han usado:

- Tecnologías para implementar el servicio web (*backend*)
- Tecnologías para el desarrollo del diseño de la aplicación (*frontend*)
- Tecnologías complementarias para facilitar el desarrollo del proyecto.

1.4.1 *Backend*

El backend de la aplicación consiste en una API REST implementada con tecnologías J2EE usando las herramientas y software que se van a explicar a continuación:

Java

Java [8] es uno de los lenguajes de programación más usados, se puede encontrar en cualquier dispositivo, desde los más cotidianos hasta servidores, videojuegos, etc. Uno de los motivos por los que ha sido usado para este proyecto es que, dado que lleva desde el año 1995

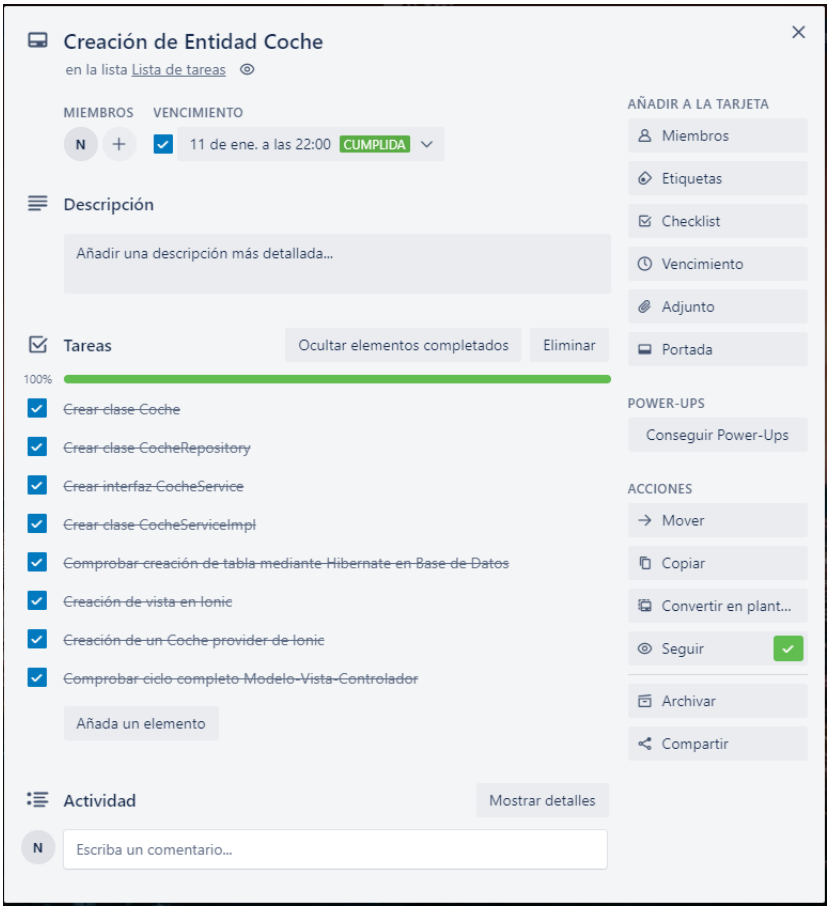


Figura 1.5: Creación de entidad coche - Tarea detallada

usándose y por lo tanto existe documentación para solventar los problemas que puedan aparecer. Además, su lenguaje es independiente de la plataforma, es decir, la aplicación de gestión que se va a desarrollar se podrá usar en cualquier dispositivo móvil independientemente del sistema operativo que use.

Hibernate

Se trata de un Framework de Java [9] cuyo objetivo es agilizar la relación entre la aplicación y la base de datos, gracias a esto se optimiza el flujo de trabajo ya que evita la creación de código repetitivo.

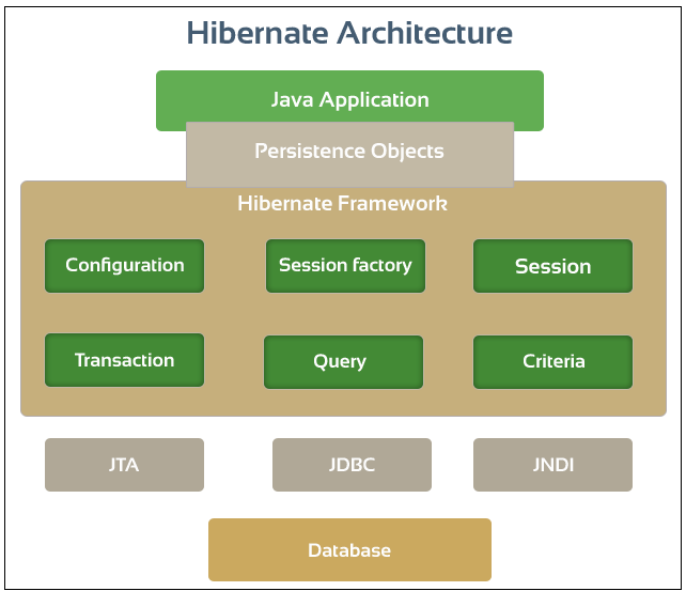


Figura 1.6: Hibernate Framework

Spring

Spring [10] también es un Framework de Java que facilita el desarrollo del código. Lo que le hace diferente es que genera código de forma automática para tareas comunes como el acceso a la base de datos. Además reduce las acciones a realizar en la configuración inicial de un proyecto. Se ha elegido para el desarrollo de este proyecto ya que la filosofía de este framework se basa en reducir al mínimo los pasos a realizar en la configuración inicial del proyecto además de su gran foco en la seguridad.

PostgreSQL

PostreSQL es un sistema de gestión de bases de datos relacionales. Es considerado el motor de BBDD más avanzado ya que, además de ser libre y gratuito, ofrece una gran cantidad de opciones avanzadas como agregar funciones personalizadas desarrolladas en lenguajes como C/C++ o Java. También permite integrarse en un sistema distribuido formado, por ejemplo, por una BBDD Oracle, una cola de mensajes además de la propia BBDD PostgreSQL. Por todos estos motivos se ha considerado una buena idea su uso en el desarrollo actual.

Maven

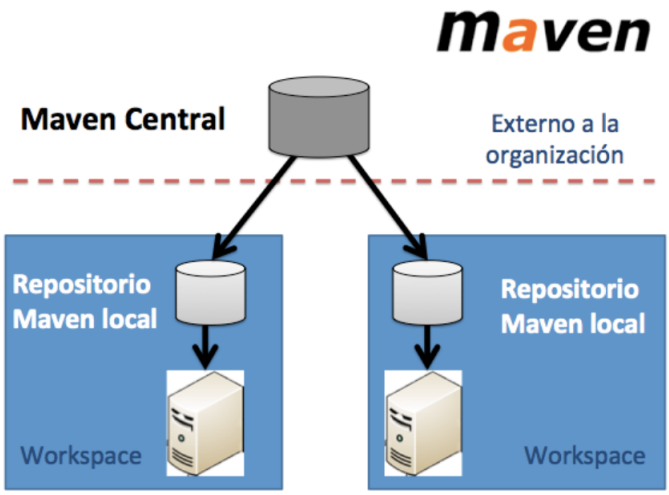


Figura 1.7: Maven

En un proyecto como el actual donde se usan frameworks como Spring o Hibernate aparece la necesidad de usar Maven ya que se usan diferentes librerías ubicadas en diferentes lugares según su utilidad y en las que hay que tener en cuenta su versión y las funciones propias de cada actualización. Además, existen librerías con dependencias de otras para su correcto funcionamiento. Gracias a Maven, todos estos inconvenientes desaparecen ejecutando el comando *mvn install* para realizar el proceso build.

REST

Se trata de un enfoque de desarrollo de proyectos y servicios web que consiste en un conjunto de restricciones con las que podemos crear un estilo de arquitectura software para desarrollar aplicaciones web que respeten HTTP. Las principales operaciones que nos va a permitir manejar son GET (leer y consultar), POST (crear), PUT (editar) y DELETE (eliminar).

Apache Tomcat

Tomcat es un servidor web, es decir, un software que se ejecuta en un servidor. Su función se basa en establecer la conexión entre un servidor y los navegadores desde los que se accede al sitio web. Al ser un software multiplataforma permite su uso tanto en servidores Unix como en Windows. Uno de los motivos de su elección para la aplicación de este desarrollo es su código abierto y gratuito ya que al tratarse de un proyecto sin financiación, el código abierto es la mejor opción para abaratar costes.

1.4.2 Frontend

En esta sección se explican las tecnologías y herramientas usadas en la implementación del *frontend*.

HTML5

HyperText Markup Lanuage [11] son las siglas que hacen referencia al lenguaje de marcado estándar más usado en los sitios web. Se usa para definir la estructura del contenido de páginas web y aplicaciones así como los enlaces que conectan las páginas web entre sí.

Ionic

Ionic [12] es un framework de desarrollo de aplicaciones móviles HTML5 para construir aplicaciones móviles híbridas. Las aplicaciones híbridas son aplicaciones móviles desarrolladas en un lenguaje de programación web como HTML5 que permiten adaptar la vista web a cualquier vista de un dispositivo móvil. Esto permite a los usuarios usar la aplicación en cualquier dispositivo sin sacrificar funcionalidades, motivo más que suficiente por el que se ha elegido para el desarrollo de este proyecto.

TypeScript

Es un lenguaje de programación gratuito de código abierto [13] y que se compila en Javascript [14] nativo. Agrega escritura estática y objetos basados en clases extendiendo la sintaxis de JavaScript y permitiendo su uso en cualquier proyecto donde éste se use. Al ser un lenguaje compilado, evita muchos errores de ejecución agilizando tiempos y rendimiento.

1.4.3 Tecnologías complementarias

Eclipse

Eclipse [15] es una plataforma de desarrollo diseñada para ser extendida a través de plugins. Es un entorno de desarrollo genérico, no está enfocado a ningún lenguaje en concreto

aunque tiene un gran público en la comunidad de Java. Esta herramienta es familiar para la alumna por su uso a lo largo del grado y por eso se ha querido usar para esta aplicación.

pgAdmin

pgAdmin [16] es una aplicación para el diseño y la administración de bases de datos PostgreSQL. Su interfaz presenta y facilita la administración de la base de datos además de incluir un editor de SQL.

Visual Studio Code

Visual Studio Code [17] es un editor de código fuente que permite trabajar con varios lenguajes de programación. Permite la descarga de extensiones con las que personalizarla a nuestras necesidades además de atajos en el teclado y refactorización de código.

Trello

Trello [7] es una herramienta de organización de tareas en un tablero. Este tablero se compone de diferentes listas en las cuales podemos ubicar nuestras tareas según su estado (*Pendiente, En curso, Finalizada o Cancelada*). Es una herramienta con la que, de manera muy visual, se puede saber el estado del sprint actual y los bloqueos existentes y por este motivo se ha escogido para el proyecto.

Git

Git [18] es un software de control de versiones, esto es, gestiona los cambios que se realizan sobre los elementos de un software, producto, etc. Fue creado para mantener de forma eficiente las diferentes versiones de las aplicaciones cuando tiene un gran número de archivos de código fuente.

Capítulo 2

Análisis y planificación

En este capítulo se describen los diferentes roles de usuario que tendrán acceso al sistema, los requisitos globales del mismo y la planificación seguida. La toma de requisitos se ha realizado mediante historias de usuario siguiendo el método Scrum.

2.1 Roles

El acceso a la aplicación móvil se restringe según el rol del usuario identificado en cada momento y para eso se diferencian tres roles: usuario sin autenticar, usuario autenticado y administrador.

- **Usuario sin autenticar**. Los usuario que no inicien sesión en el sistema únicamente tendrán acceso a la funcionalidad de iniciar sesión, recuperar contraseña si ya están registrados o darse de alta.

- **Usuario autenticado**. Cuando un usuario se identifica con usuario y contraseña podrá dar de alta nuevos coches, ver la información de éstos y de los existentes, así como solicitar las citas que desee para cada uno de ellos. También podrá personalizar el idioma de la aplicación eligiendo entre español e inglés y contactar con el taller mediante el formulario disponible en cualquier momento. Además, tendrá acceso a las promociones existentes en el taller obteniendo la posibilidad de realizar la reserva con un descuento. Por último, el usuario puede ver el proceso de su cita y descargar la factura cuando esta sea creada en el sistema.

- **Administrador**. El usuario administrador podrá gestionar a los usuarios pudiendo eliminarlos cuando lo desee así como cambiar el estado de las citas para que el usuario autenticado pueda verlo. También podrá crear nuevas facturas sobre las citas existentes y eliminar estas citas si lo cree conveniente.

2.2 Product Backlog

En esta sección se muestran las historias de usuario que conforman el *Product Backlog* de la aplicación.

Historia de usuario	Nombre	Puntuación
US01	Registro	3
US02	Autenticación	3
US03	Recuperar contraseña	3
US04	Cerrar sesión	1
US05	Eliminar usuarios	2
US06	Añadir coche	2
US07	Visualizar coche	1
US08	Eliminar coche	2
US09	Añadir cita	3
US10	Mostrar cita	1
US11	Filtrar citas	2
US12	Modificar estado cita y notificación	2
US13	Eliminar cita	2
US14	Alta de factura	3
US15	Visualizar factura	4
US16	Aplicar promociones	3
US17	Visualizar perfil	2
US18	Contactar con taller	3
US19	Cambiar idioma	3

Cuadro 2.1: Product Backlog

2.3 Funcionalidades

US01 - Registro

Para darse de alta en el sistema, un usuario nuevo debe proporcionar un nombre de usuario único, un correo electrónico y una contraseña. El sistema comprobará la validez de los datos introducidos y en caso de ser correctos, se creará una nueva cuenta de usuario. En caso contrario, se mostrará un mensaje por pantalla indicando el error. Cuando el registro haya sido exitoso, el usuario ya tendrá acceso a todas las funcionalidades de la aplicación disponibles para usuarios autenticados.

US02 - Autenticación

Para acceder al sistema, el usuario debe introducir su usuario y contraseña. Si éstos son correctos, estará autenticado en el sistema. En caso contrario, se mostrará un mensaje de error. En cuanto la autenticación sea exitosa, el usuario podrá acceder a las funcionalidades para usuarios autenticados.

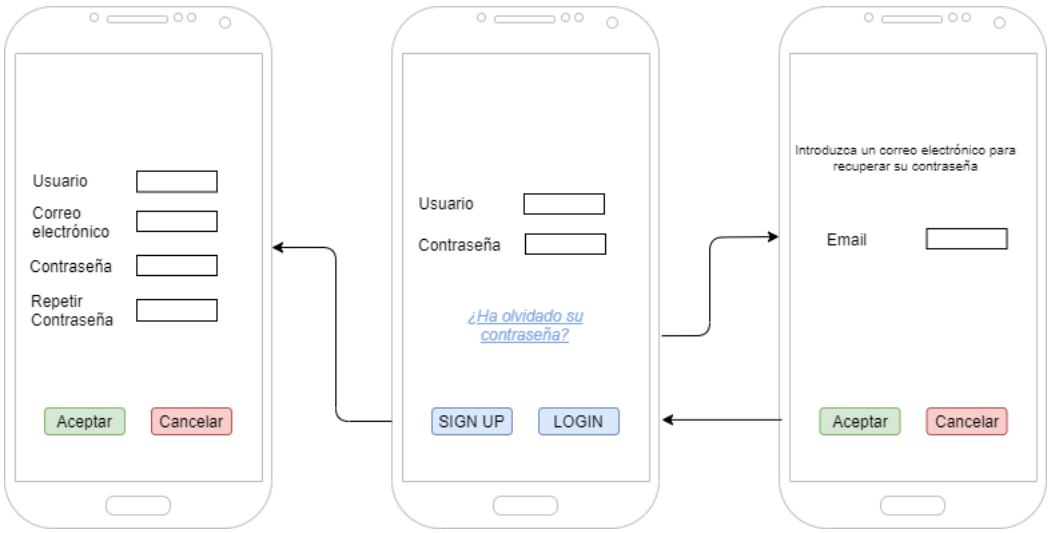


Figura 2.1: Funcionalidades de autenticación de usuarios

US03 - Recuperar contraseña

Un usuario registrado puede recuperar su contraseña usando el mismo correo electrónico con el que se dio de alta en el sistema.

US04 - Cerrar sesión

Los usuarios autenticados tienen la posibilidad de cerrar la sesión en cualquier momento pulsando el botón *Cerrar sesión*. En cuanto se realiza esa acción, el usuario solo podrá acceder a las funcionalidades de usuarios no autenticados.

US05 - Eliminar usuarios

Los usuarios administradores podrán eliminar los usuarios y toda su información desde la vista **Usuarios.**

US06 - Añadir coche

Los usuarios autenticados podrán dar de alta sus coches en la aplicación. Para realizar esta acción, el sistema mostrará un formulario para rellenar en el que se pedirá la siguiente información:

- Marca del vehículo.
- Matricula
- Kilometraje.
- Año de fabricación.
- Combustible que usa el vehículo.
- Color.
- Potencia.

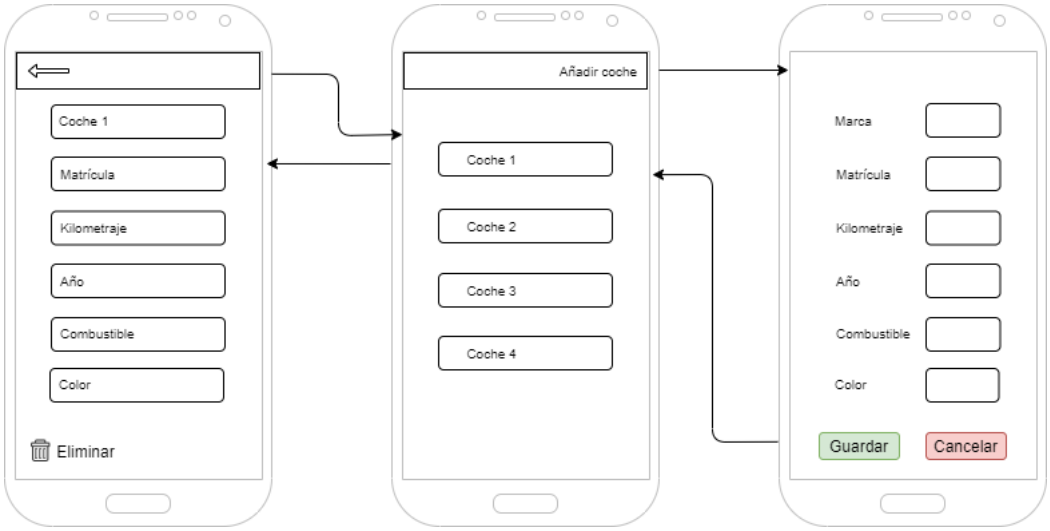


Figura 2.2: Gestión de vehículos

US07 - Visualizar coche

El usuario tendrá disponible una vista de todos sus coches dados de alta en la aplicación, donde podrá pulsar en cada uno de ellos y será redirigido a otra pantalla en la que se muestra una vista de detalle con toda la información introducida en el momento del registro. El usuario administrador tendrá disponible una vista donde se muestra la lista de citas en estado *Pendiente,* es decir, los coches que van a llegar al taller o los que han llegado ya pero aún no se ha iniciado su reparación.

US08 - Eliminar coche

Desde la vista de detalle de cada coche, el usuario tendrá disponible la opción de eliminar el vehículo siempre que lo desee. Una vez el vehículo sea eliminado, no se podrá recuperar su información. En caso de querer visualizarlo de nuevo se tendría que proceder al registro de un nuevo vehículo (US05).

US09 - Añadir cita

Los usuarios autenticados podrán añadir una cita para llevar sus vehículos al taller cuando estimen necesario. Los datos que deberán introducir en el formulario son los siguientes (ver figura 2.3):

- Vehículo para la cita.
- Fecha.
- Hora.
- Tipo de cita (reparación o mantenimiento).

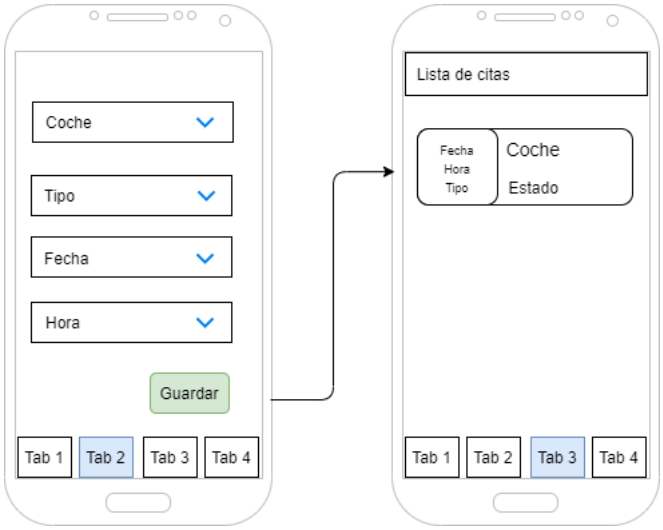


Figura 2.3: Añadir cita

US10 - Mostrar cita

El usuario autenticado tendrá disponible una vista de las citas que haya concertado donde visualizará la fecha, hora y tipo de cita así como el estado actual de la misma.

US11 - Filtrar citas

Los usuarios podrán filtrar las citas que visualizan usando dos criterios: el estado de la cita y el tipo (mantenimiento o reparación). Para ello, en la vista de **Lista de citas** encontrará un botón que lo redirigirá a otra vista donde se especifican los criterios. Además, cuenta con la posibilidad de limpiar los filtros cuando lo desee.

US12 - Modificar estado cita y notificación

El usuario administrador tendrá la posibilidad de cambiar el estado de la cita para que el usuario esté al tanto de los cambios y pueda saber cuando ha finalizado el proceso y puede recoger su coche.

US13 - Eliminar cita

El usuario administrador, desde la misma vista en la que modifica el estado de la cita, podrá eliminarla pulsando en el icono correspondiente.

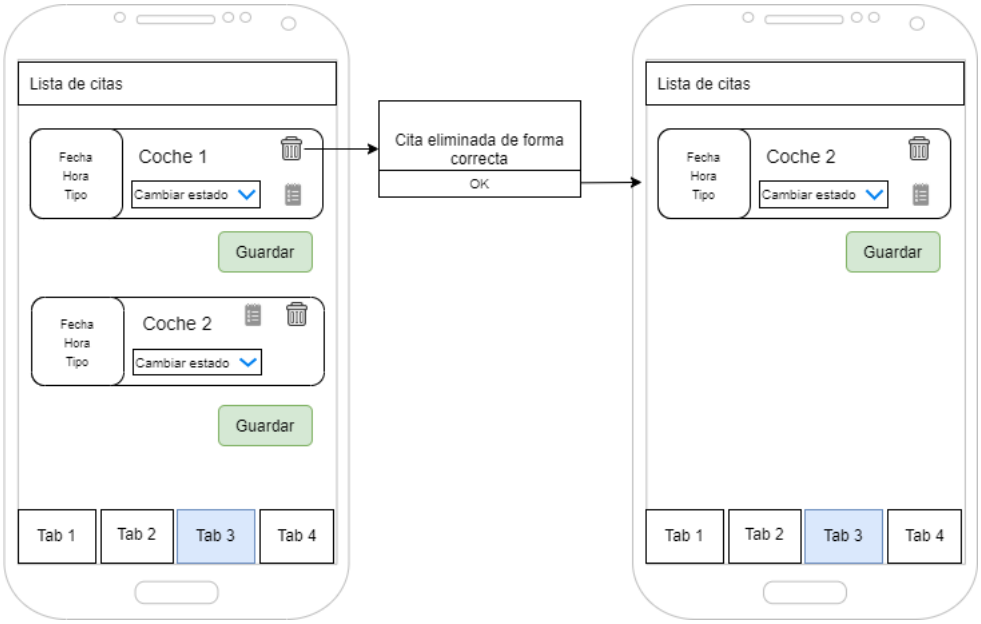


Figura 2.4: Eliminar Cita

US14 - Alta factura

El usuario adminsitrador, una vez la cita haya pasado al estado de *Finalizado*, podrá dar de alta la factura con la cuantía que el usuario deberá abonar. Desde la vista **Lista de citas** tendrá un botón habilitado que le redirigirá a una nueva vista para que introduzca los campos correspondientes.

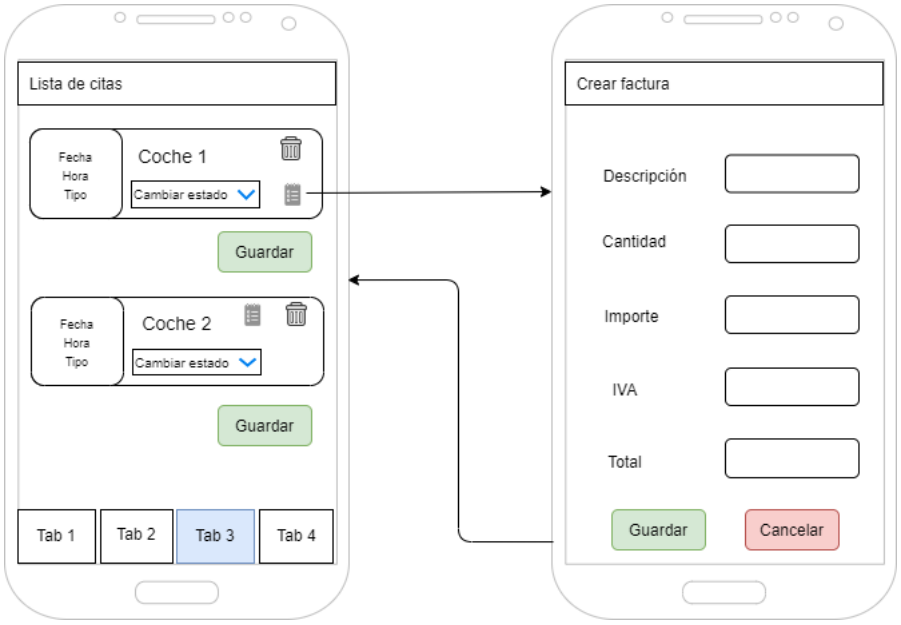


Figura 2.5: Alta Factura

US15 - Visualizar factura

Desde la vista de **Lista de citas** el usuario autenticado tendrá disponible la opción *Descargar factura* desde la cual se descargará un PDF con la información de la misma cuando ya esté disponible.

US16 - Aplicar promociones

Desde la vista **Perfil** el usuario podrá visualizar las promociones que tiene activas el taller. El usuario podrá pulsar en la que más le convenga y la aplicación le redirigirá a la vista de *Añadir cita* donde podrán crear la cita con los criterios que deseen y además, podrán visualizar el descuento que van a obtener. Este descuento quedará reflejado en la factura que el cliente descarga una vez ha finalizado todo el proceso de reparación.

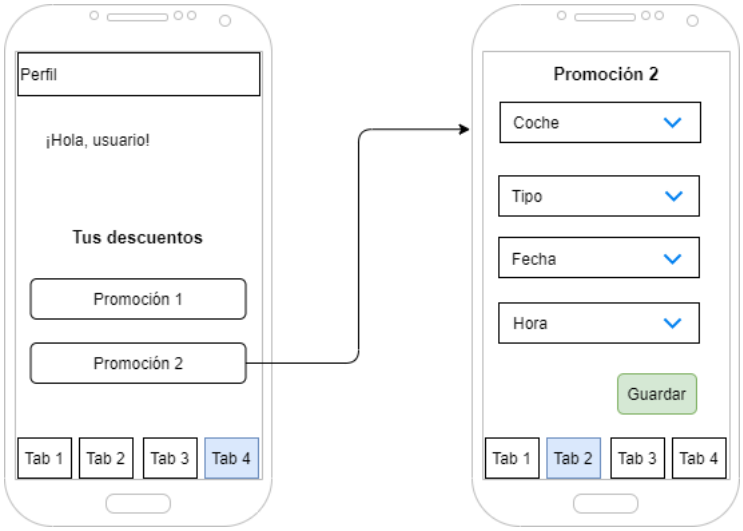


Figura 2.6: Cita con promoción

US17 - Visualizar perfil

Desde la vista **Perfil** el usuario podrá ver su información, así como las promociones activas en el momento de visualización dentro del taller.

US18 - Contactar con taller

Desde el menú lateral, el usuario autenticado podrá acceder a un formulario desde el cual se podrá poner en contacto con el taller para reportar problemas, consultar dudas o lo que desee. En dicho formulario se pedirá la siguiente información:

- Nombre y apellidos.
- Correo electrónico en el que se desee recibir la respuesta.
- Contenido del mensaje.

US19 - Cambiar idioma

Desde el menú lateral, un usuario autenticado tendrá la opción de cambiar el idioma de la aplicación. Podrá elegir entre Inglés y Español. Por defecto, el idioma será el mismo que el del dispositivo desde el que se acceda al sistema.

2.4 Sprints

Esta sección está destinada a mostrar la planificación llevada a cabo a lo largo del proyecto en el cual se siguió la metodología *Scrum*.

Sprint 0: Formación tecnológica y creación del proyecto base

En la primera iteración de este proyecto se han expuesto las ideas iniciales para desarrollar así como los objetivos que nos gustaría cumplir. También se ha diseñado un arquetipo sobre el que construir la aplicación. Además, se ha destinado tiempo de este Sprint a formación en las tecnologías usadas.

Sprint 1: Gestión de usuarios

En esta iteración se han desarrollado las funcionalidades referentes a registrar y autenticar usuarios, así como recuperar contraseña y desconectar sesión.

- US01 - Registro
- US02 - Autenticación
- US03 - Recuperar contraseña
- US04 - Cerrar sesión
- US05 - Eliminar usuarios

Sprint 2: Gestión de coches

El propósito de este sprint es la implementación de las funcionalidades que giran entorno a la gestión de un coche:

- US06 - Añadir coche
- US07 - Visualizar coche
- US08 - Eliminar coche

Sprint 3: Gestión de citas

Una vez implantadas las funcionalidades referentes a los vehículos, se aborda la gestión de las citas reflejadas en las siguientes historias de usuario:

- US09 - Añadir cita
- US10 - Mostrar cita
- US11 - Filtrar citas
- US12 - Modificar estado cita y notificación
- US13 - Eliminar cita

Sprint 4: Gestión de factura

Se implementan las funcionalidades relacionadas con el rol administrador y el rol usuario autenticado para la correcta visualización de la factura y la aplicación de promociones sobre la misma:

- US14 - Alta factura
- US15 - Visualizar factura
- US16 - Aplicar promociones

Sprint 5: Componentes extra

Se ha decidido añadir algunas funcionalidades extra a la aplicación como cambiar el idioma del sistema, visualizar la información del perfil del usuario y contactar con el taller mecánico con el propósito de mejorar la experiencia del usuario:

- US17 - Visualizar perfil
- US18 - Contactar con taller
- US19 - Cambiar idioma

Sprint 6: Memoria y mejoras

En la última iteración del proyecto se desarrolla la memoria actual. Por otro lado, se incluyen pequeñas mejor en la aplicación surgidas de la revisión de los sprints anteriores.

2.5 Planificación temporal

Según las directrices de Scrum, la planificación del proyecto se ha llevado a cabo tratando de forma independiente cada *sprint*. Además, todos los sprints deberían tener una duración similar. Sin embargo, por el desconocimiento de las tecnologías usadas no ha sido posible realizar una estimación de los primeros sprints de forma exacto. Sumado a esto, la situación laboral de la alumna ha imposibilitado la dedicación a tiempo completo en el proyecto y ha supuesto algunos atrasos en ciertos puntos de la aplicación. En la figura 2.7 se muestra la planificación final del proyecto con un diagrama de Gantt [19].

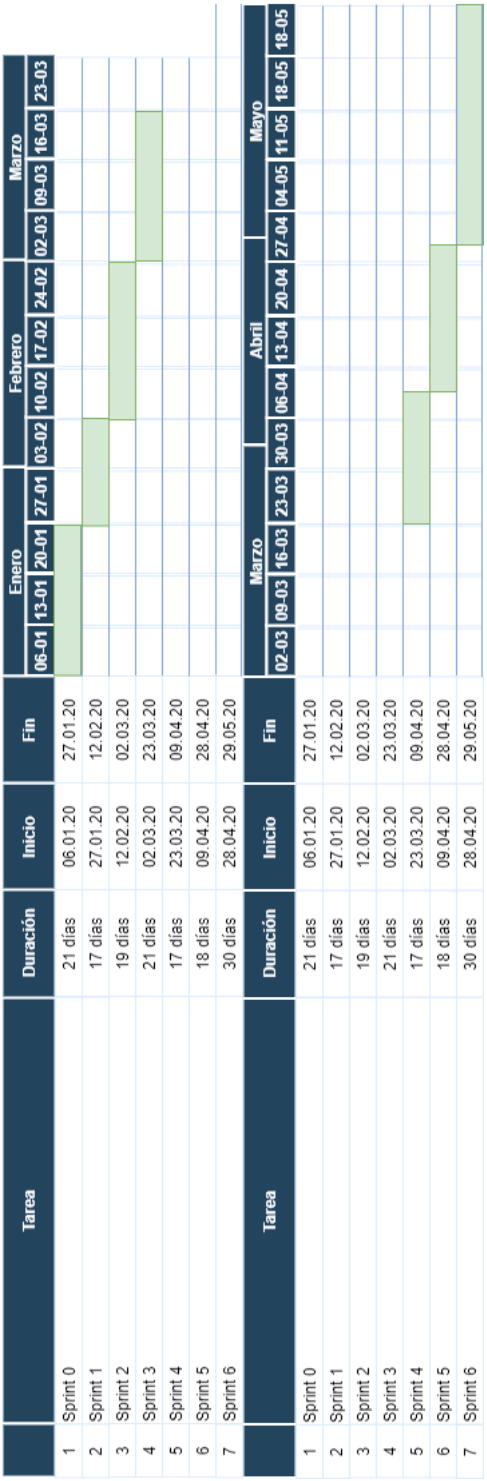
2.6 Cálculo de costes

Para la realización del cálculo aproximado de los costes de este proyecto se ha tenido en cuenta las horas invertidas en el desarrollo del mismo. Además, en cuanto a material, el coste no se ha calculado ya que se ha usado el portátil personal de la alumna, pero en un desarrollo real habría que considerar no sólo el valor del equipo sino su amortización a lo largo de la vida del proyecto. Teniendo en cuenta todo lo nombrado y las horas dedicadas que se van a especificar en la siguiente tabla, se ha establecido un coste de 30€/hora que el desarrollador percibirá como salario.

Sprint	Nombre	Inicio	Fin	Horas
0	Proyecto base	06/01/2020	27/01/2020	105
1	Gestión de usuarios	27/01/2020	12/02/2020	85
2	Gestión de vehículos	12/02/2020	02/03/2020	95
3	Gestión de citas	02/03/2020	23/03/2020	105
4	Gestión de factura	23/03/2020	09/04/2020	85
5	Componentes extra	09/04/2020	28/04/2020	90
6	Memoria y mejoras	28/04/2020	29/05/2020	150
7	**TOTAL**	06/01/2020	29/05/2020	**760**

Cuadro 2.2: Planificación de tiempo dedicado al proyecto

Por tanto, el coste aproximado del proyecto sería de 750 h $\times$ 25 €/hora = **22.800 €**.



	Tarea	Duración	Inicio	Fin
1	Sprint 0	21 días	06.01.20	27.01.20
2	Sprint 1	17 días	27.01.20	12.02.20
3	Sprint 2	19 días	12.02.20	02.03.20
4	Sprint 3	21 días	02.03.20	23.03.20
5	Sprint 4	17 días	23.03.20	09.04.20
6	Sprint 5	18 días	09.04.20	28.04.20
7	Sprint 6	30 días	28.04.20	29.05.20

Figura 2.7: Planificación

Capítulo 3
Desarrollo

En este capítulo se va a explicar de forma detallada el proceso seguido para desarrollar la aplicación incluyendo aspectos tanto técnicos como funcionales de las iteraciones.

3.1 Modelo de datos

El modelo de datos (ver figura 3.1) se organiza en torno a la entidad *Coche* ya que en él se basa toda la información. Se ha diseñado en base al alcance definido en la primera reunión de Scrum y se han ido modificando detalles por las necesidades que se han encontrado en la implementación. Cada coche podrá ser dado de alta por un usuario, representado por la entidad *Cuenta* que será su único dueño, aunque ese usuario puede tener en su posesión varios coches. Por otro lado, el coche puede tener *Citas* que serán dadas de alta por el usuario y que será únicas para cada coche. Además, el coche puede contar con *Promociones* que supondrán un ahorro en la factura del usuario. Por último, cada cita generará una *Factura* única, es decir, cada cita tendrá su factura y una misma factura no puede pertenecer a varias citas.

3.2 Sprint 0: Formación tecnológica y arranque

Una vez definidas las tecnologías a usar, se procedió a realizar una formación sobre ellas comenzando por las usadas en el *backend*. El estudio se centró en Hibernate y Spring ya que en el resto de tecnologías el conocimiento era más avanzado. La mayoría del esfuerzo se invirtió en las tecnologías usadas en el *frontend* ya que la alumna no tenía muchos conocimientos sobre ellas, en concreto sobre *Typescript* y *Ionic*.

3.2.1 Creación del proyecto base

En este *Sprint* se realizó la configuración del entorno de desarrollo. Además, se implementó un arquetipo básico que proporciona las funcionalidades necesarias para arrancar las

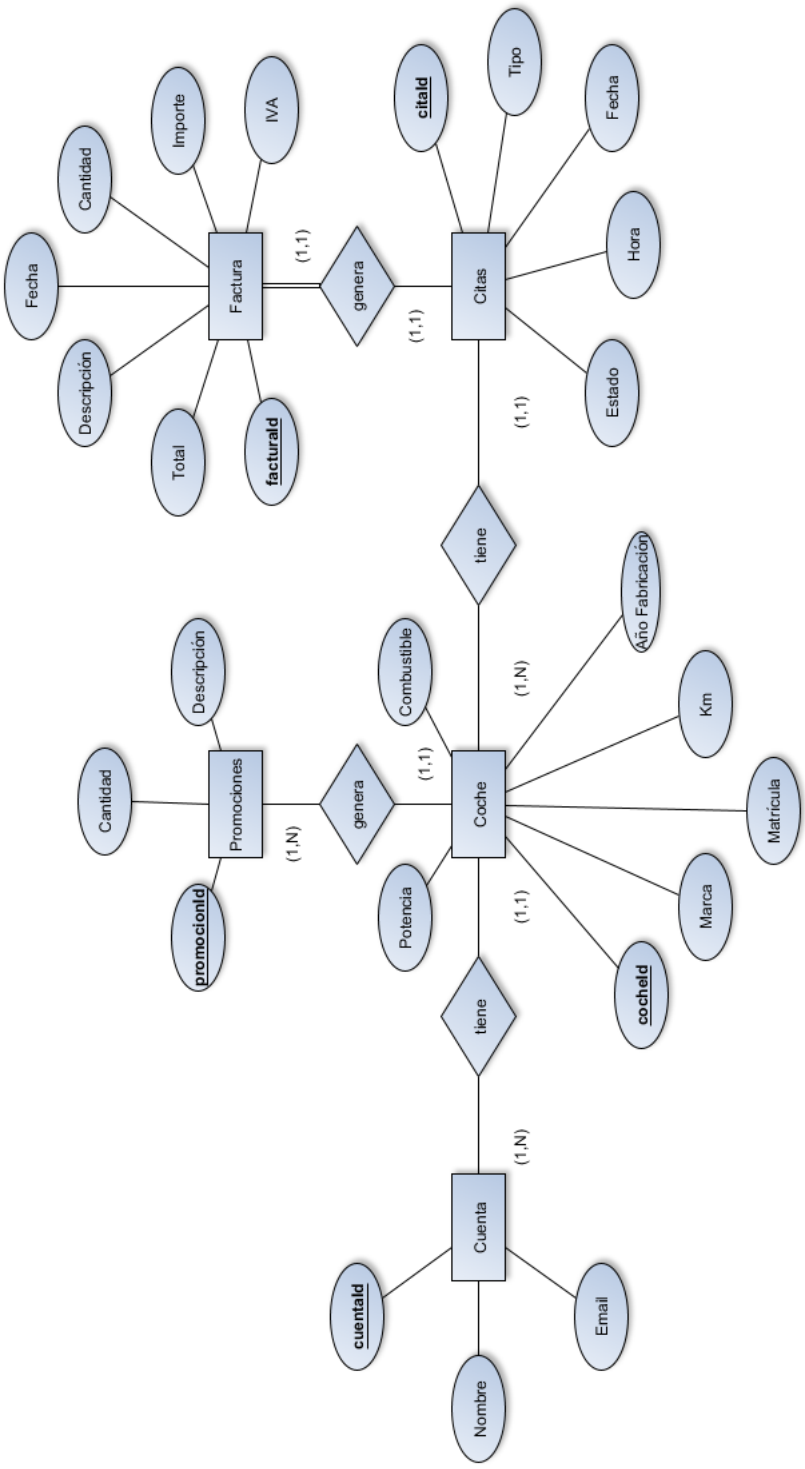


Figura 3.1: Entidad Relación

tecnologías usadas construyendo una base para la aplicación. Proporciona información sobre cómo se usa Maven para construir el proyecto y ejecutarlo con un servidor Tomcat y además, ya se implementan las primeras anotaciones de Spring que se usarán durante el desarrollo.

Estructura del *backend*

El primer paso ha sido ejecutar el arquetipo con Maven, paso tras el cual se ha generado la siguiente estructura:

- **src/main/java:** es el directorio más extenso ya que contiene el código fuente de la aplicación almacenando cada entidad en un paquete. Inicialmente, este directorio contiene el *Application.java* y el paquete *Account* conformado por los siguientes archivos:
  - *Account.java*
  - *AccountRepository.java*
  - *AccountService.java*
  - *AccountServiceImpl.java*
- **src/main/resources:** este directorio almacena el archivo *persistence.properties* en el que se encuentran los parámetros de conexión con la base de datos así como las especificaciones que indican si los datos se van a persistir o no cuando el servidor se reinicie.
- **src/test/java:** contiene los paquetes y las clases en las cuales se implementan las pruebas realizadas sobre la aplicación.

Estructura del *frontend*

Para realizar la base del proyecto de Ionic se ha usado un comando que facilita el proceso ya que genera la estructura automáticamente con las especificaciones que el usuario decida. En este caso, la aplicación, llamada *TallerApp*, se va a generar con el componente *Ionic Tabs*:

```
ionic start tallerapp tabs
```

Figura 3.2: Generación del proyecto Ionic

La estructura generada por Ionic como se puede observar en la imagen 3.3 y consta de los siguientes componentes:

- **src/app/app.component.ts**. Es el primer componente cargado en la aplicación y lo que hace es cargar otros componentes. Además, en él se define el idioma por defecto de todo el sistema.

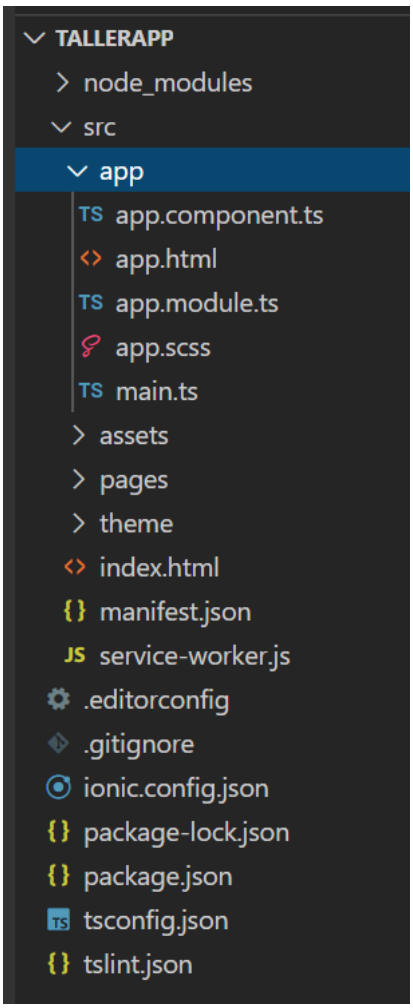


Figura 3.3: Estructura inicial Ionic

- **src/app/app.html**. Dentro de este archivo se define la plantilla para toda la aplicación que incluye el menú lateral al se podrá acceder desde las páginas principales para cerrar sesión, cambiar el idioma, etc.
- **src/app/app.module.ts.** Es el módulo principal y contiene las declaraciones de todos los componentes que se van a utilizar.
- **src/assets.** Almacena las imágenes e iconos que se muestran en la aplicación.
- **src/pages.** Se trata de un árbol de directorios en los que existe una carpeta por cada página que se muestra en la aplicación. Estos directorios están formados por los mismos archivos aunque su contenido sea diferente:
  - *archivo.html.* Contiene el código para estructurar el contenido de la página.
  - *archivo.module.ts.*
  - *archivo.scss.* En él se define el estilo de la página.
  - *archivo.ts.* Contiene los métodos usados para interaccionar con la página.
- **src/providers.** Se trata de archivos Typescript que establecen comunicación al servicio REST de la aplicación a través de peticiones HTTP.

```
deleteCar(carId: any) {

  let url = URL_SERVICIOS + "/deleteCar/" + carId;

  let headers = new Headers({
    'Content-Type': 'application/json'
  });

  headers.append('Authorization', '' + this.AuthToken);

  let options = new RequestOptions({
    headers: headers
  });

  return this.http.delete(url, options).map((res: any) => res)
}
```

Figura 3.4: Función para eliminar un coche

- **src/theme.** Almacena el fichero *variable.scss* que contiene variables globales que se van a usar para definir el estilo de la aplicación en los archivos *.scss* como colores, fuentes de texto,etc.

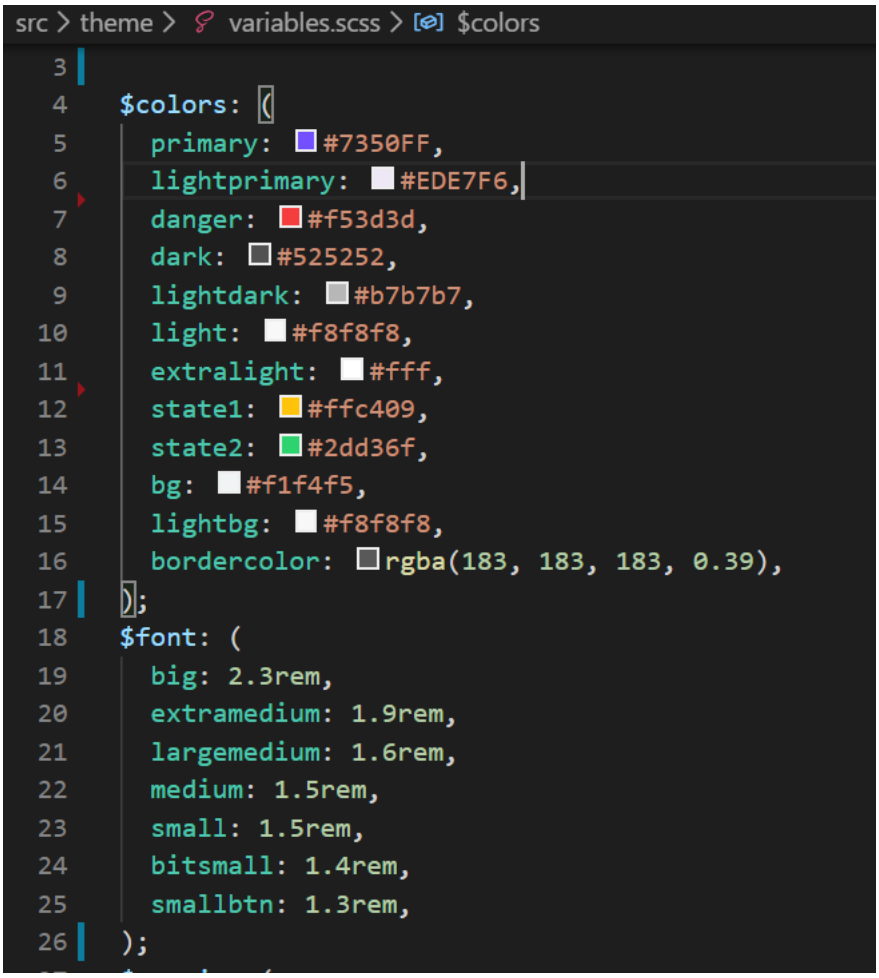


Figura 3.5: Variables para estilo

3.3 Sprint 1: Gestión de usuarios

3.3.1 Sprint Backlog

En este *Sprint* se van a desarrollar las historias de usuario relacionadas con las funcionalidades de gestión de usuarios y su autenticación. A continuación se define el *Sprint Backlog* con las tareas concretas de cada historia.

- **US01 - Registro.** Como usuario no registrado quiero registrarme en el sistema.

Tareas
Crear una cuenta en Firebase para realizar el servicio de autenticación
Implementar la funcionalidad en el servicio REST para almacenar los usuarios en el servicio
Crear la vista de *Registro* para recoger las credenciales del usuario
Realizar pruebas para verificar el correcto funcionamiento del método

Cuadro 3.1: Sprint 1. Registro

- **US02 - Autenticación.** Como usuario no autenticado quiero iniciar sesión en la aplicación

Tareas
Creación y almacenamiento en local de un token cuando el usuario se autentica en el sistema
Validación del sistema de tokens en el servicio usando Firebase
Crear la vista de *Autenticación*
Realizar pruebas para verificar el correcto funcionamiento de los métodos implementados

Cuadro 3.2: Sprint 1. Autenticación

- **US03 - Recuperar contraseña.** Como usuario registrado quiero recuperar mi contraseña en caso de olvidarla.

Tareas
Creación de la vista para recuperar la contraseña con los métodos que proporciona Firebase
Validación las funcionalidades implementadas

Cuadro 3.3: Sprint 1. Recuperar contraseña

- **US04 - Cerrar sesión.** Como usuario autenticado quiero cerrar sesión desde cualquier página de la aplicación.

Tareas
Crear una nueva vista en el menú lateral para la funcionalidad de cerrar sesión
Validación del correcto funcionamiento de los métodos implementados

Cuadro 3.4: Sprint 1. Cerrar sesión

- **US05 - Eliminar usuarios.** Como usuario administrador puedo eliminar los usuarios y su información.

Tareas
Desarrollo del método en el *backend*
Crear una nueva vista para la funcionalidad con los métodos de Firebase
Validación el correcto funcionamiento de los métodos implementados

Cuadro 3.5: Sprint 1. Eliminar usuarios

3.3.2 Diseño e implementación

- **US01 - Registro**
  Firebase nos permite implementar la autenticación de la aplicación a través de un correo electrónico y una contraseña aportando seguridad ya que, al almacenar los datos de autenticación en la nube en lugar de usar un servidor propio existe menos riesgo de péridida. En Typescript se encuentra el elemento *Promises* con el cual se pueden realizar tareas como recibir una respuesta de una petición HTTP a través de funciones que se ejecutará hasta que la promesa se resuelva o se rechace. Para indicar que la función ha concluido, existen dos resultados:

  – *Resolve*. Función para resolver la promesa de forma exitosa.

  – *Reject*. Función para rechazar la promesa.

  Estas funciones se han usado en la implementación de un método para el registro de los usuarios, el cual almacenará las credenciales del usuario en Firebase y cuyo control de errores se realizará de la siguiente forma: en caso de éxito, se enviará una petición POST para que el API REST reciba los datos del formulario de inicio de sesión de la aplicación. En caso contrario, si la solicitud devuelve un resultado erróneo por algún

motivo de fallo de conexión con Firebase o con el *backend*, se mostrará una alerta por pantalla.

```
registerUser(email: string,password: string, nombre: string){
    return this.anFrAuth.auth.createUserWithEmailAndPassword(email,
    password)
        .then(function(user) {
            user.updateProfile({
                displayName: nombre
                }).then(function() {
                    //Actualización exitosa
            }, function(error) {
                let alerta = this.alertCtrl.create({
                    title: 'Error',
                    subTitle: error,
                    buttons: ['OK']
                });
                alerta.present();
            });
        })
        .catch(err => Promise.reject(err))
}
```

Listing 3.1: Método para el registro de usuarios

```
signup() {
    this.accountService.signup(this.user.email, this.user.nombre)
}
```

Listing 3.2: Método de almacenamiento de las credenciales en la BD

```
signupFireBase() {
    this.auth.registerUser(this.user.email, this.user.password,
    this.user.nombre)
        .then((user) => {
            this.signup();
            this,navCtrl.setRoot(LoginPage);
        })
        .catch(err => {
        let alert = this.alertCtrl.create({
            title: 'Error',
            subTitle: err.message,
            buttons: ['OK']
        });
        alert.present();
    })
```

```
15 }
```

Listing 3.3: Creación el usuario en Firebase

En el *frontend* se ha implementado un formulario en la página **SignUp** para que el usuario introduzca sus credenciales usando los componentes de Ionic *FormGroup* y *FormControl*, como se puede observar en la figura 3.6. Dicho formulario contiene una validación de los campos introducidos en la que se establecen los criterios de los campos. En este caso, tanto el campo nombre como email y contraseña son campos obligatorios y además, la contraseña tiene un mínimo de longitud para garantizar su seguridad. Además, en esta vista se informa al usuario de que sus datos serán tratados siguiendo la ley de protección de datos.

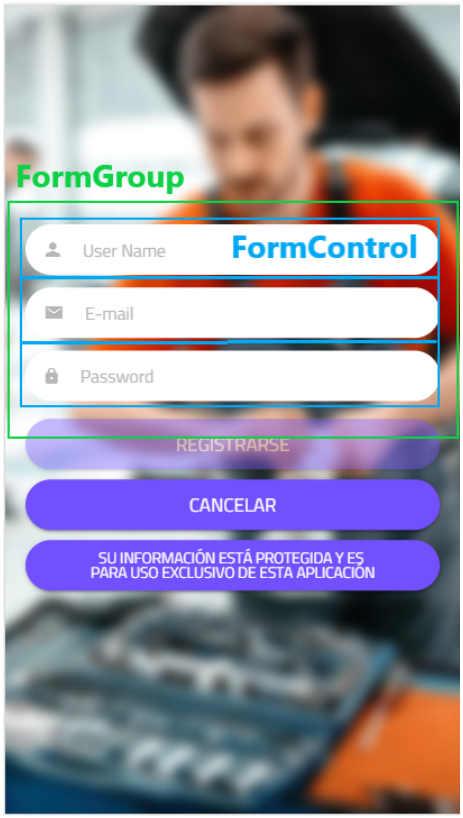


Figura 3.6: Formulario para registrarse

- **US02 - Autenticación**
  Para realizar la autenticación de usuarios ya registrados se han seguido los estándares basados en *Json Web Token* [20] para la creación de tokens con caducidad, que indican si la sesión sigue activa o no, por lo que no es necesario comprobar si el usuario está logeado en cada operación. *Firebase Authentication UI* aplica el funcionamiento de estos

tokens, siendo el propio usuario el que tenga en su almacenamiento local el token de autenticación que se enviará en cada petición para ser validado por el servicio. En el *frontend* el usuario tendrá un formulario en la página de **Login** (ver figura 3.7) en el que introducirá su nombre de usuario y contraseña, siendo estos campos obligatorios para iniciar sesión.

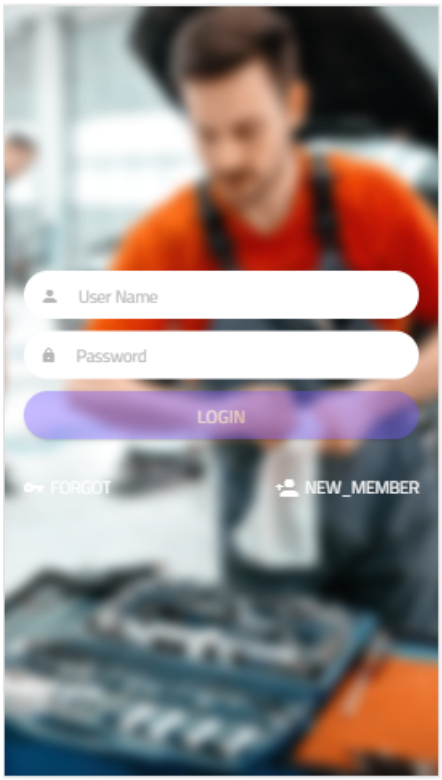


Figura 3.7: Formulario para iniciar sesión

- **US03 - Recuperar contraseña**
  Firebase ofrece el método `firebase.auth.sendPasswordResetEmail(email)` con el que los usuarios registrados en el sistema pueden resetear su contraseña en caso de haberla olvidado. Se envía un correo con un link que redirigirá al usuario a un formulario en el que se introducirá la nueva contraseña, la cual será actualizada en el sistema permitiendo al usuario autenticarse.

```
resetPassword(email:string) {
    var auth = firebase.auth();
    auth.sendPasswordResetEmail(email).then(function() {
        //Email enviado
    }).catch(function(error) {
        let alerta = this.alertCtrl.create({
            title:'Error',
            subTitle: error.message,
            buttons: ['Ok']
        });
    alert.present();
    });
    this.navCtrl.setRoot(LoginPage);
}
```

Listing 3.4: Función para recuperar la contraseña

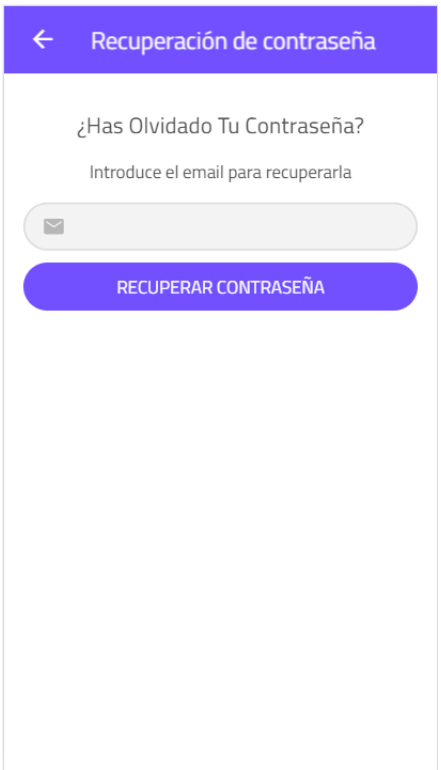


Figura 3.8: Vista de recuperación de contraseña

- **US04 - Cerrar sesión**
  Para cerrar la sesión de la aplicación se usará el método `firebase.auth().signOut()` que deshabilitará y eliminará el token del almacenamiento local al quedar inutilizable.

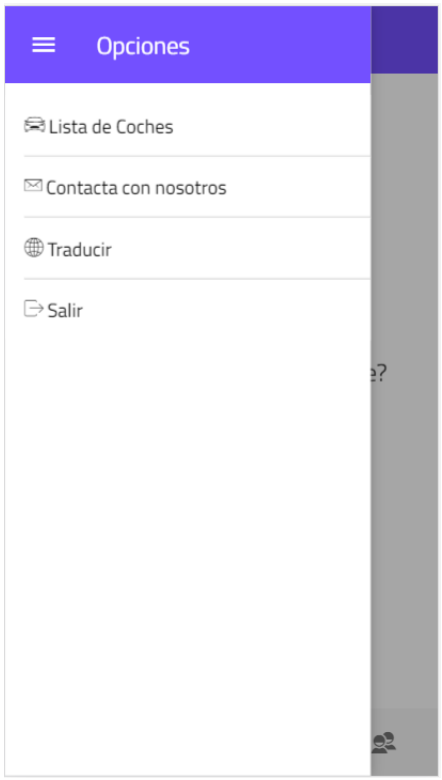


Figura 3.9: Cerrar sesión

- **US05 - Eliminar usuarios**

  El administrador del sistema, como parte de sus funciones de gestión de usuario, tiene la posibilidad de eliminar los usuarios con toda su información cuando lo desee. Para esto, se ha usar el SDK de administración que permite interactuar con Firebase para ejecutar acciones privilegiadas. Si la eliminación del usuario se realiza de forma correcta en Firebase, se enviará una petición DELETE al *backend* para que se elimine toda la información almacenada del usuario en la base de datos de la aplicación.

```
deleteUserFirebase(email:string){
    admin.auth().getUserByEmail(email)
        .then(function(userRecord){
            admin.auth().deleteUser(userRecord.uid)
                .then(function())}
      .catch(function(error) {
            logger.info(error);
        });
}
```

Listing 3.5: Función para eliminar usuario de Firebase

3.4 Sprint 2: Gestión de coches

3.4.1 Sprint Backlog

- **US06 - Añadir coche.** Como usuario autenticado quiero añadir nuevos coches.

Tareas
Implementar la entidad Coche en el *backend*
Crear un formulario para recibir la información del coche
Crear una validación del formulario
Validar el correcto funcionamiento de los métodos

Cuadro 3.6: Sprint 2. Añadir coche

- **US07 - Visualizar coche.** Como usuario autenticado quiero visualizar mis coches.

Tareas
Implementar las funciones necesarias para obtener la información del coche en el *backend*
Crear una vista para mostrar los datos del coche
Validar el correcto funcionamiento de los métodos

Cuadro 3.7: Sprint 2. Visualizar coche

- **US08 - Eliminar coche.** Como usuario autenticado quiero eliminar coches de mi propiedad.

Tareas
Implementar las funciones necesarias para eliminar la información del coche en el *backend*
Añadir en la vista del coche la opción de eliminarlo
Validar el correcto funcionamiento de los métodos implementados.

Cuadro 3.8: Sprint 2. Eliminar coche

3.4.2 Diseño e implementación

- **US06 - Añadir coche**
  Durante el desarrollo de esta historia de usuario se ha creado la relación entre el coche y el usuario que lo ha creado con la finalidad de que éste pueda gestionarlos. Al intentar crear la relación bidireccional *Cuenta - Coche* se ha detectado un error de recursividad infinita ya que en Hibernate se debe especificar el límite de cada relación. Si esto no se especifica, surgirá el problema ya que un coche tiene un usuario asociado, el cual tiene más coches asociados. Para solventarlo, existen las siguientes anotaciones que se han incluído en el código del *backend*:

  - **Json Managed Reference.** Indica que es la parte ”padre” del enlace bidireccional.
  - **Json Back Reference**. Indica que es la parte ”hijo” del enlace.

```
@OneToMany(mappedBy="cuenta",fetch = FetchType.EAGER)
@JsonManagedReference (value="cuenta - coche")
private List<Coche> cocheList;

```

Listing 3.6: Relación Cuenta-Coche - Cuenta.java

```
@ManyToOne
@JoinColumn(name = "cuentaId")
@JsonBackReference(value="cuenta - coche")
private Cuenta cuenta;

```

Listing 3.7: Relación Cuenta-Coche - Coche.java

En la vista de coches, a la cual se puede acceder desde el menú inferior, se ha creado un formulario para dar de alta los vehículos. La primera vez que se realiza esta operación,

al no haber ningún coche dado de alta, se mostrará un botón en el centro de la pantalla, el cual redirigirá al usuario al formulario donde rellenará los datos (ver figura3.10).

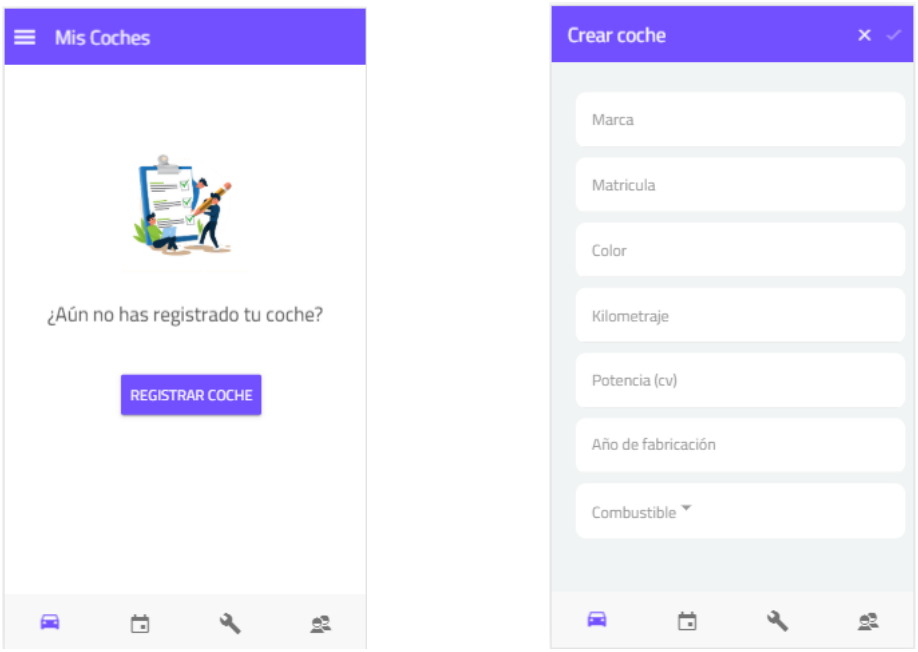


Figura 3.10: Alta de vehículos

Cada campo tiene unos criterios de validación adaptados al contenido que se espera, por ejemplo, no está permitido que en el campo de *potencia* se introduzcan letras. Además, para una mayor seguridad de que no se van a introducir valores erróneos o vacíos, el sistema no permitirá añadir el vehículo a la base de datos hasta que todos los campos estén rellenados de forma correcta, como se puede ver en la imagen 3.11.

- **US07 - Visualizar coche**
  Para visualizar la lista de los coches que el usuario ha dado de alta en su cuenta, se ha implementado una petición GET en el *backend*, la cual obtiene de la base de datos el objeto *Coche* con toda la información disponible. Para estos datos, que se van a mostrar por pantalla se ha creado la página de **Coche** y **Detalle de Coche** en el *frontend*. En la primera vista se muestra la lista de `ion-card` de vehículos del usuario las cuales contienen el acceso al detalle (ver figura 3.12) en el que se puede observar toda la información almacenada de la entidad *coche*.

  Para el administrador, se usará la misma vista que en el caso del usuario autenticado. En este caso, se visualizarán los vehículos de cualquier usuario cuyo estado de la cita sea *Pendiente*.

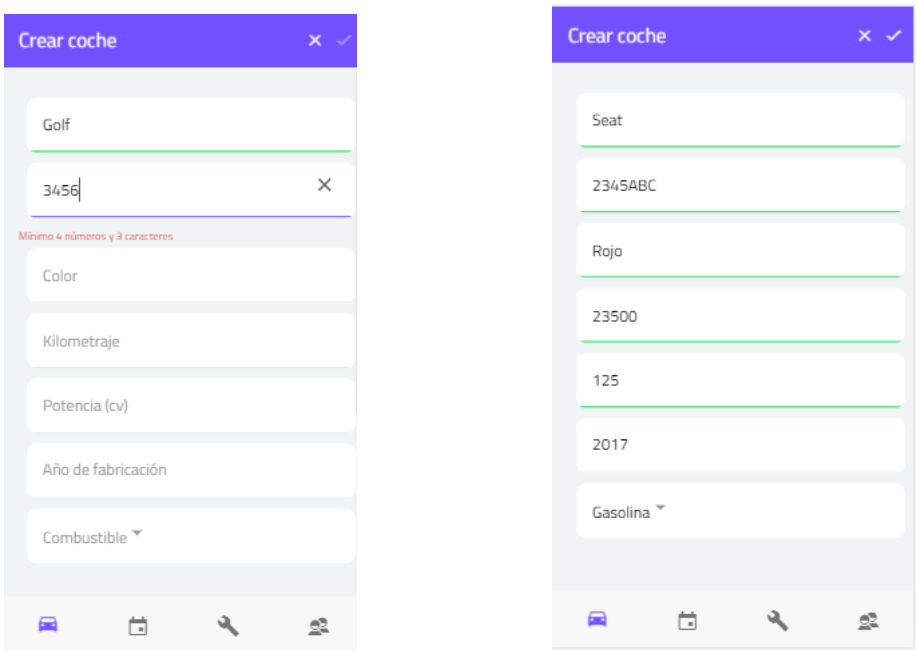


Figura 3.11: Validación de formulario

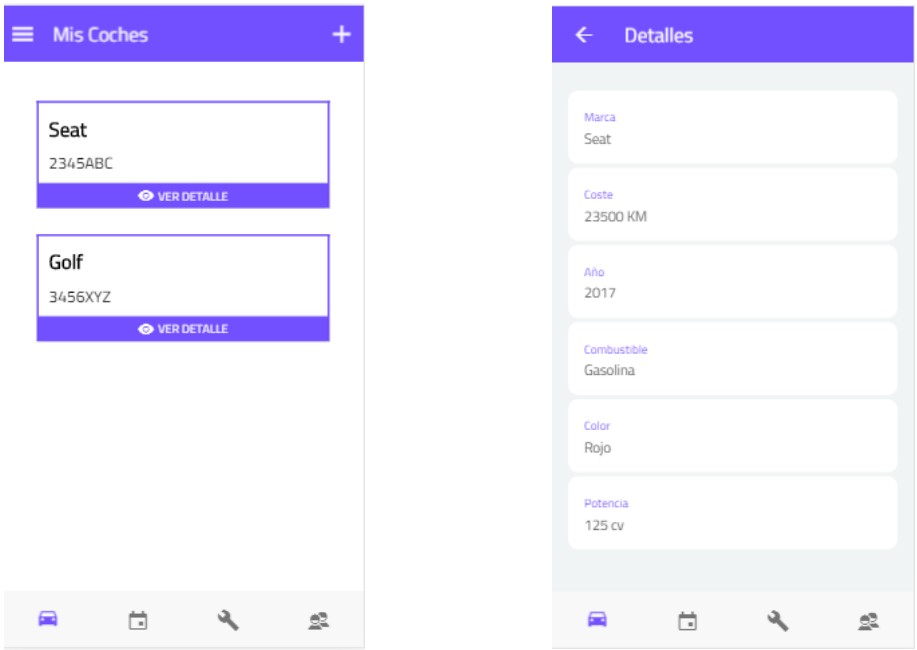


Figura 3.12: Vista de coche

- **US08 - Eliminar coche**
  Si en algún momento un usuario quisiese eliminar un coche del sistema (porque lo ha vendido, ya no lo usa, etc), siempre y cuando no exista ninguna cita con un estado diferente a *Finalizada* o *Cancelada*, podrá hacerlo desde la vista de **Lista de Coches**. Para ello desde el *frontend* se ha implementado un `ion-item-sliding` con el que, deslizando el `ion-card` del coche deseado a la izquierda, se muestra la opción de eliminarlo como se ve en la imagen 3.13. Esto enviará una petición de DELETE al *backend*, el cual va a eliminar la información del vehículo y de sus componentes (citas, facturas, etc) de la base de datos.

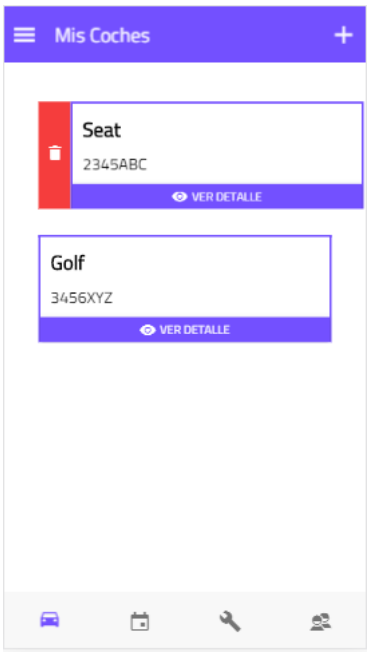


Figura 3.13: Eliminar vehículos

3.5 Sprint 3: Gestión de citas

3.5.1 Sprint Backlog

- **US09 - Añadir cita.** Como usuario autenticado y con vehículos registrados, quiero concertar una cita.

Tareas
Implementar la entidad Cita en el *backend*
Crear un formulario para recibir la información de la cita
Validar el correcto funcionamiento de los métodos

Cuadro 3.9: Sprint 3. Añadir cita

- **US10 - Mostrar cita.** Como usuario autenticado quiero visualizar mis citas y que se me notifique de los cambios sobre ellas.

Tareas
Implementar las funciones necesarias para obtener la información de la cita en el *backend*
Crear una vista para mostrar los datos de la cita
Validar el correcto funcionamiento de los métodos

Cuadro 3.10: Sprint 3. Mostrar cita

- **US11 - Filtrar citas**. Como usuario autenticado independientemente del rol que tenga quiero filtrar mis citas.

Tareas
Implementar los métodos necesarios para filtrar las citas en el *frontend*
Implementar un método para borrar los filtros aplicados previamente
Crear una vista para seleccionar los filtros
Validar el correcto funcionamiento de los filtros

Cuadro 3.11: Sprint 3. Filtrar citas

- **US12 - Modificar estado cita y notificación.** Como usuario administrador quiero modificar el estado de la cita.

Tareas
Implementar los métodos necesarios para modificar el campo estado en el *backend*
Modificar la vista de *Lista de citas* para que el administrador pueda modificar el estado
Implementar los métodos necesarios para las notificaciones
Validar el correcto funcionamiento de los filtros

Cuadro 3.12: Sprint 3. Modificar estado cita y notificación

- **US13 - Eliminar cita.** Como usurario administrador quiero eliminar citas.

Tareas
Implementar las funciones necesarias para eliminar la información de la cita en el *backend*
Modificar la vista de *Lista de citas* para que el administrador pueda eliminar la cita
Validar el correcto funcionamiento de los métodos implementados.

Cuadro 3.13: Sprint 3. Eliminar cita

3.5.2 Diseño e implementación

- **US09 - Añadir cita**
  Para añadir una cita ha sido necesario establecer la relación *Coche - Cita* teniendo en cuenta la recursividad infinita que ya se ha explicado en el sprint anterior.

```
1 @OneToMany(mappedBy="coche",fetch = FetchType.EAGER)
2 @JsonManagedReference (value="coche - cita")
3 private List<Cita> citaList;
4
```

Listing 3.8: Relación Coche-Cita - Coche.java

```
@ManyToOne
@JoinColumn(name = "cocheId")
@JsonBackReference(value="coche - cita")
private Coche coche;

```

Listing 3.9: Relación Coche-Cita - Cita.java

Para la recogida de datos que el usuario introduce, se ha creado un formulario en el que existen dos tipos de objetos de Ionic, `ion-select` con `ion-option` usado para introducir el coche y el tipo de arreglo que necesita el vehículo (reparación o mantenimiento). Por otro lado, se usa `ion-datetime` tanto para el día como la hora de la cita. Los selectores usados evitan la inserción de valores erróneos en la base de datos al mostrar unas opciones limitadas. Además, se ha creado un enlace desde la vista de **Lista de citas** cuando no exista ninguna cita creada que redirigirá al usuario a la vista de **Crear cita** (ver figura 3.14).

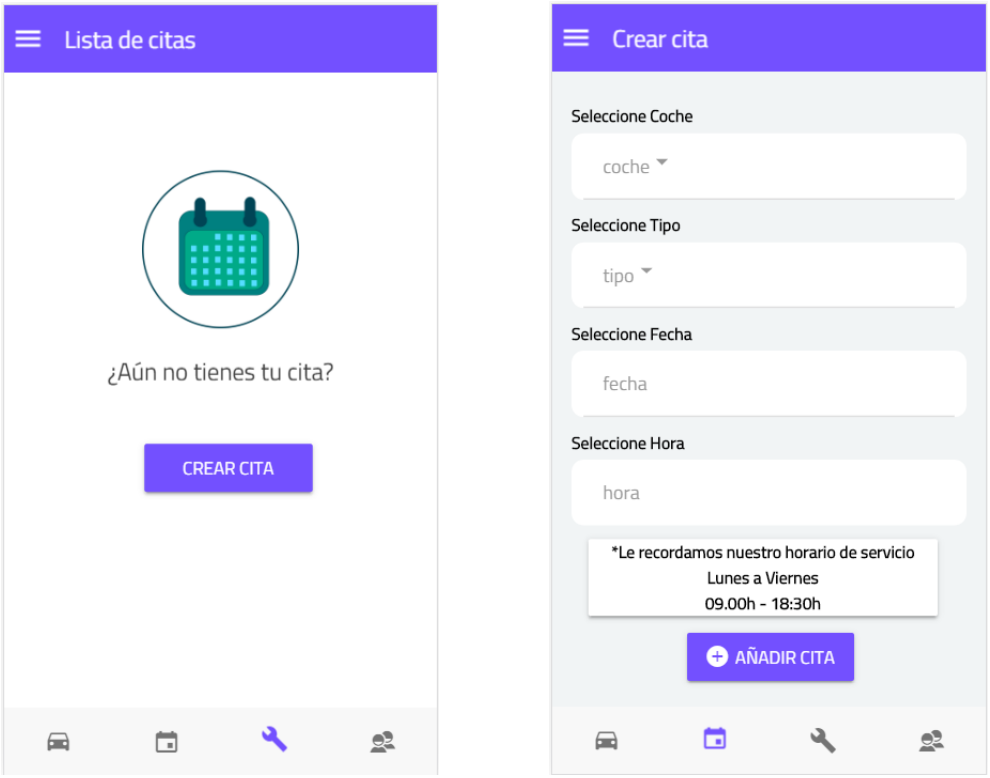


Figura 3.14: Vista de añadir cita

- **US10 - Mostrar cita**
  Para mostrar la lista de citas de los coches de cada usuario, se implementa una petición GET que recupera toda la información de la entidad *cita* de la base de datos. Una vez recuperado, en el *frontend* se visualiza dentro de la vista **Lista de citas** donde el usuario visualizará la fecha y la hora de dichas citas (ver figura3.15).
  Además, el usuario podrá llevar un control del estado en el que están estas citas según los cambios que el administrador vaya haciendo. Los estado que puede tomar una cita son: *Pendiente*, *En curso*, *Finalizada* o *Cancelada*.

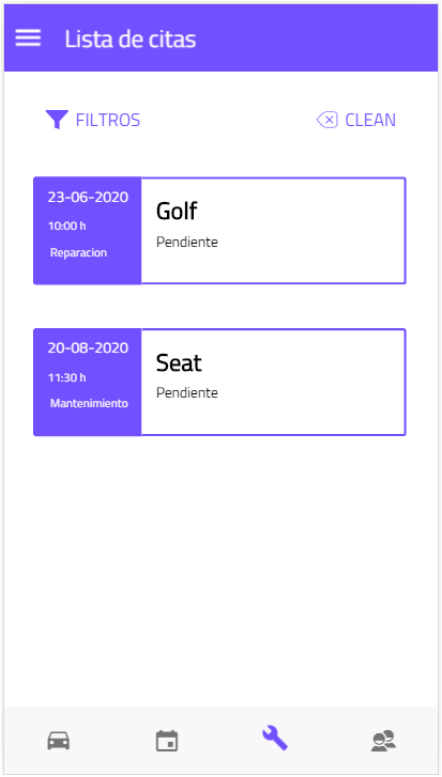


Figura 3.15: Lista de citas

- **US11 - Filtrar citas**
  Para esta funcionalidad no es necesario realizar ningún desarrollo en el *backend*, siendo todos los cambios e implementaciones dentro del *frontend*.
  Dentro de la vista **Lista de citas**, se ha implementado un método que permite filtrar las citas que se visualizan según su estado o la tipología. Estas características van a ser especificadas dentro de una nueva vista (ver figura 3.16) a la que se accede desde un `ion-button` en la parte superior de la pantalla de citas. Además, existe la posibilidad de eliminar todos los filtros que se hayan aplicado usando el `ion-button` de limpiar filtros que reseteará los valores de los campos *tipo* y *estado*.

Usando la clase `NavCtrl` [21] que ofrece Ionic, se envían los parámetros entre las páginas que se especifiquen. Una vez el usuario se encuentra en la vista de **Filtros** y ha elegido los valores, los aceptará, acción que activará la función `navCtrl.pop()`. El usuario volverá a la vista de **Lista de citas** que habrá recibido un `events.subscribe()` para modificar los datos que se visualizan según los criterios previamente especificados.

```
cleanFilters() {
this.tipo = null;
this.state = null;
this.initializeCars();
}
```

Listing 3.10: Reseteo de los filtros

```
apply(){
this.events.publish('state:tipo', this.state, this.tipo);
this.navCtrl.pop();
}
```

Listing 3.11: Envío de los valores en los campos a filtrar y vuelta a la lista de citas

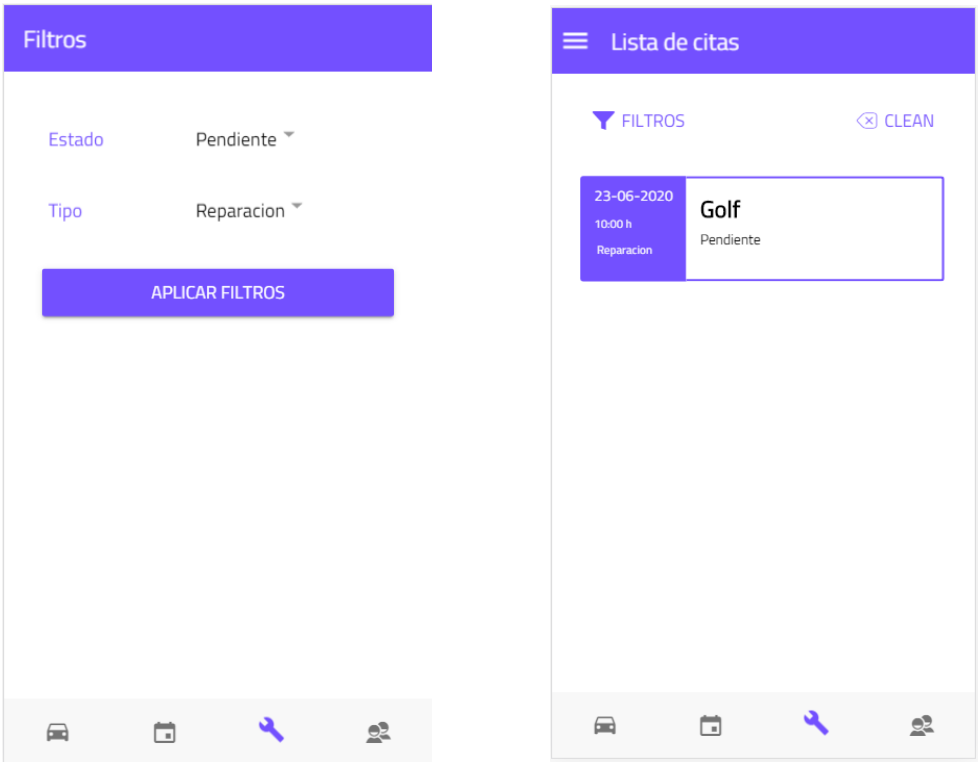


Figura 3.16: Aplicación de filtros

- **US12 - Modificar estado cita y notificación**
  La actualización del estado de la cita únicamente puede realizarla el usuario administrador. Para ello cuenta con un desplegable en el que selecciona el valor que debe tomar la cita en cada momento (ver figura 3.17). Esto enviará una petición POST al backend donde se realizará una operación set. Se busca la cita indicada a través de su *id*, y si esta efectivamente existe, se actualizará su estado al indicado por el administrador.

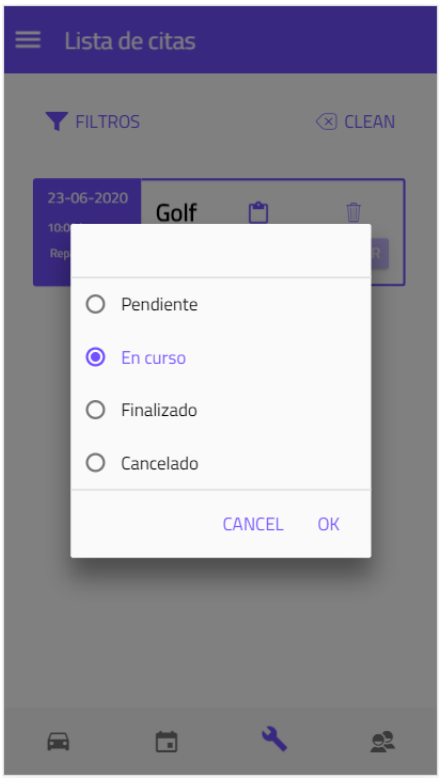


Figura 3.17: Cambio de Estado - Vista Administrador

Cuando el usuario acceda a su sesión de la aplicación, podrá observar los cambios realizados por el administrador (ver figura 3.18). Además de esto, en el momento en el que el administrador haga estas actualizaciones en la cita, se enviará una notificación push al dispositivo agilizando así la comunicación *Taller-Cliente*. Para gestionar estas notificaciones se usa el método de Firebase `admin.messaging().send(message)` que a través del SDK de Firebase Admin permite enviar un mensaje al dispositivo del usuario informándole de que se han realizado cambios en sus citas.

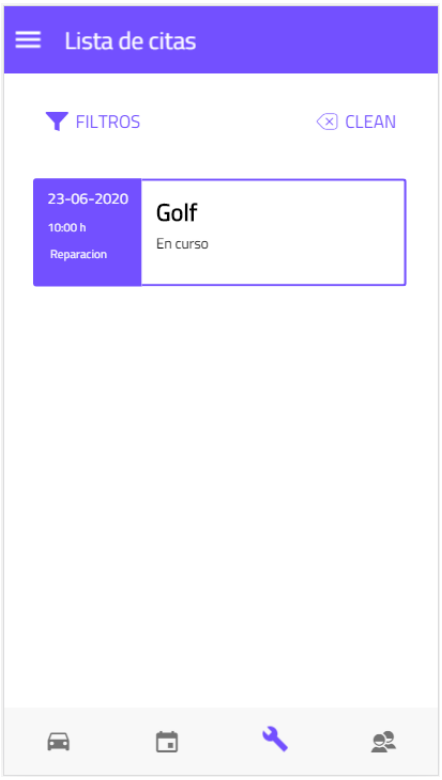


Figura 3.18: Estado Cambiado - Vista Usuario

- **US13 - Eliminar cita**
  El usuario administrador tiene la posibilidad de eliminar una cita (ver figura 3.19) aunque exista la posibilidad de cambiar el estado de una cita a *Cancelada*. Para ello, se envía una petición DELETE al *backend* eliminando toda la información existente en la base de datos como por ejemplo, la factura asociada a la cita en caso de existir.

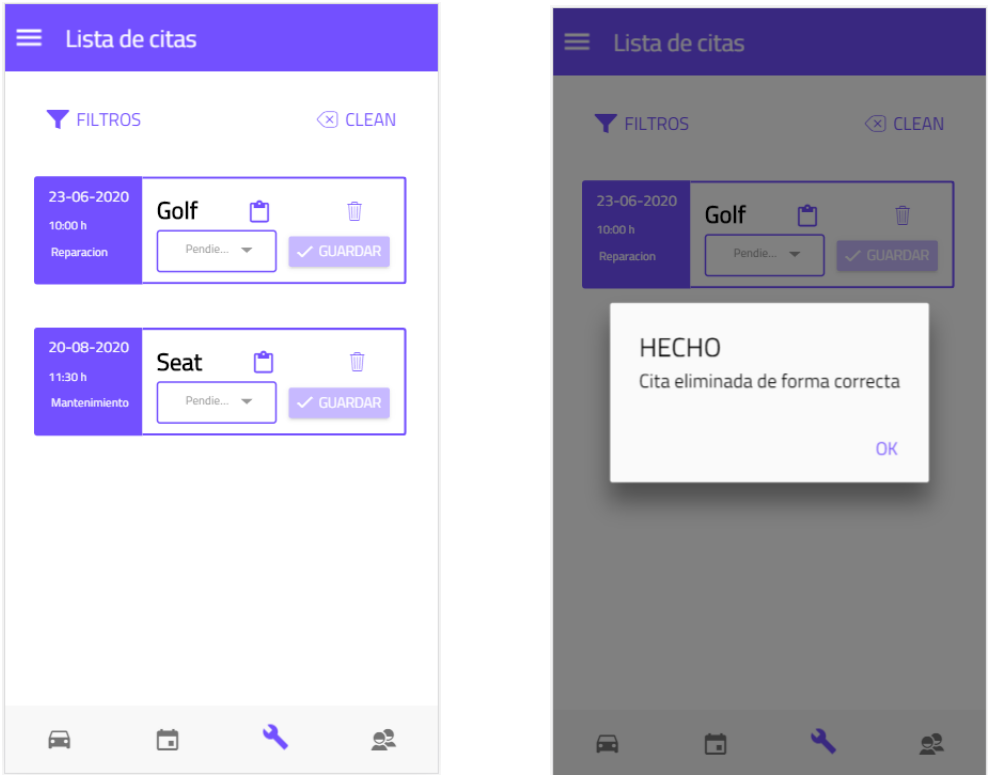


Figura 3.19: Eliminar Cita

3.6 Sprint 4: Gestión de factura

- **US14 - Alta de factura**

Tareas
Implementar la entidad Factura *backend*
Crear una vista para dar de alta los datos de la *factura*
Validar el correcto funcionamiento de los métodos implementados

Cuadro 3.14: Sprint 4. Alta de factura

- **US15 - Visualizar factura**

Tareas
Implementar las funciones necesarias para eliminar la información de la factura en el *backend*
Habilitar la opción de descargar la factura en el *frontend*
Validar el correcto funcionamiento de los métodos implementados

Cuadro 3.15: Sprint 4. Visualizar factura

- **US16 - Aplicar promociones**

Tareas
Implementar las funciones necesarias aplicar promociones en el *backend*
Modificar la vista de *Profile* para ofrecer al usuario las promociones
Modificar la vista de *Crear cita* aplicar los descuentos a las citas correspondientes
Validar el correcto funcionamiento de los métodos implementados

Cuadro 3.16: Sprint 4. Aplicar promociones

3.6.1 Diseño e implementación

- **US14 - Alta de factura**
  Una vez la cita se actualiza al estado *Finalizado*, se puede proceder a dar de alta la factura. Para ello, el administrador contará con un `ion-button` que le conducirá a una nueva vista en la que podrá introducir los campos necesarios para generar una factura.

Por ejemplo, en la imagen 3.20 se ha realizado el cambio de 4 ruedas y cada rueda tiene un coste de 100€. Esto enviará una petición POST al *backend* donde ya se ha creado la relación *Cita - Factura*. Una vez los datos se ha dado de alta la factura con los campos requeridos, se calcula en el *backend* el importe total de la factura con los impuestos. Este cálculo ha de realizarse de forma interna en la aplicación para evitar la generación de errores por introducir datos erróneos.

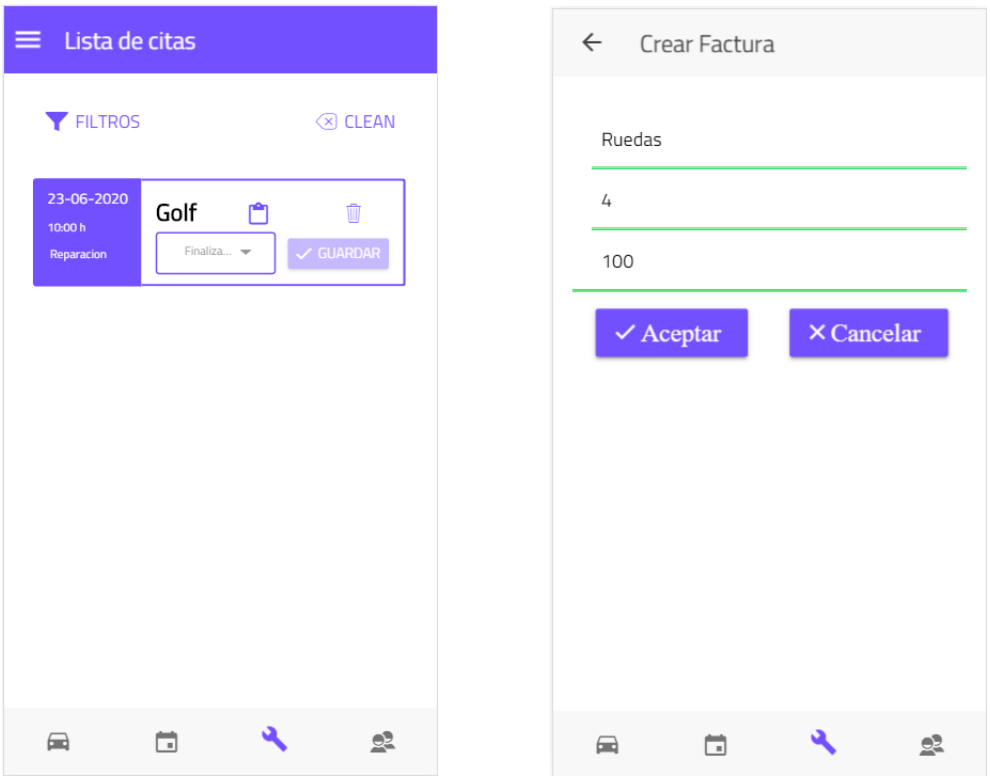


Figura 3.20: Alta Factura

- **US15 - Visualizar factura**
  Una vez el administrador ha dado de alta la factura, el usuario tendrá disponible la opción de descargar la factura en su dispositivo en formato PDF (ver figura 3.21). Para esto, se realiza una petición GET al *backend* para obtener los datos necesarios para construir el PDF. En el *frontend* se crea un método en el que se usa la función `docDefinition` para crear el documento con el formato deseado (columnas, datos empresariales, datos del cliente...). Cuando el usuario descargue el PDF en su dispositivo, se generará el documento con los datos pertinentes y se abrirá con el visualizador que nativo existente.

```
    downloadPdf(appointment: any) {
    this.createPdf(appointment.bill) //cuando clica en descargar,
    primero general el Pdf
    if (this.plt.is('cordova')) {
      this.pdfObj.open();
    } else {
      // En un navegador se realiza la descarga
      this.pdfObj.download();
      //download es un método predefinido de la librería para pdfs
    de Ionic
    }
  }

```

Listing 3.12: Descarga de Factura

Figura 3.21: Descargar factura - Vista Usuario

- **US16 - Aplicar promociones**
  En la vista *Perfil* el usuario tendrá disponibles las promociones existentes en su taller mecánico. En el *frontend* estas promociones están creadas sobre `ion-list` a través de las cuales se redirige al usuario a la vista de **Crear Cita** en la que se especifica con qué descuento contará al crear la cita.

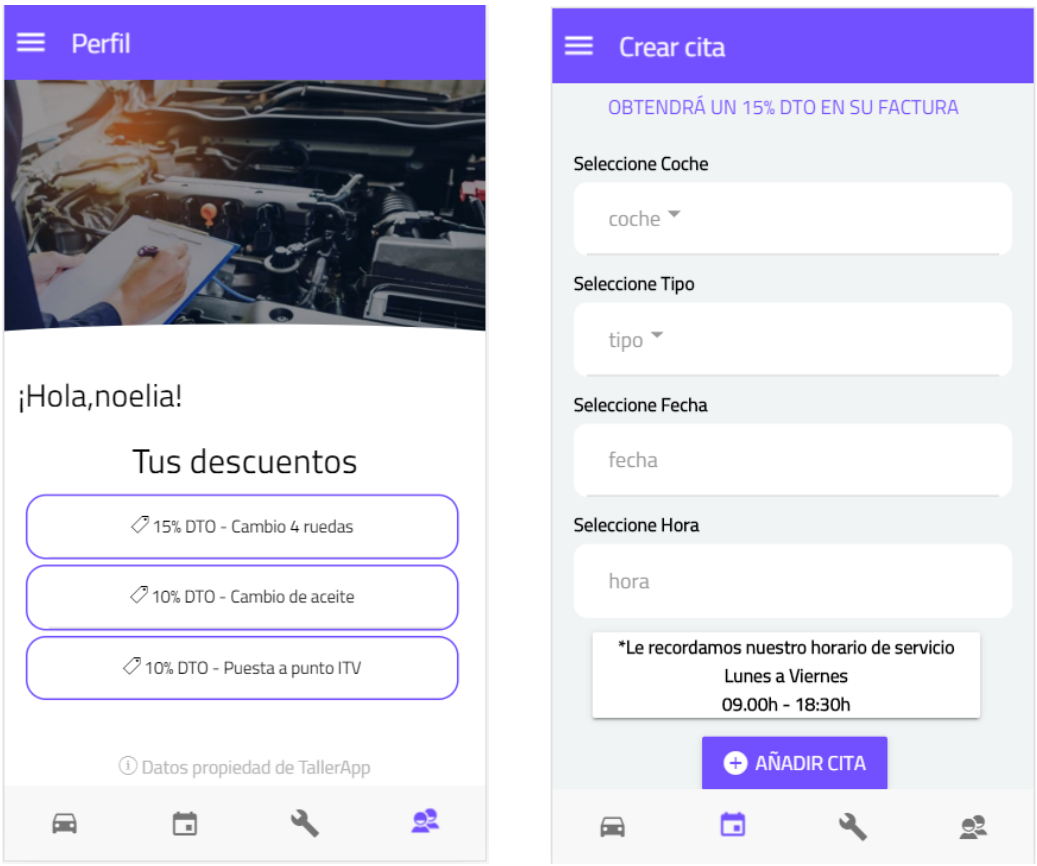


Figura 3.22: Promociones

3.7 Sprint 5: Componentes extra

En esta sección, se han implementado algunas funcionalidades extra en la aplicación para mejorar la experiencia del usuario.

- **US17 - Visualizar perfil.** Como usuario autenticado quiero ver mi información de usuario

Tareas
Implementar un método GET para obtener la información del usuario del servicio REST
Crear una vista para visualizar la información del usuario

Cuadro 3.17: Sprint 5. Visualizar perfil

- **US18 - Contactar con taller**. Como usuario autenticado quiero poder contactar con el taller mecánico.

Tareas
Crear la vista de *Contacta con nosotras* para ofrecer al usuario la posibilidad de dejar un mensaje al taller
Configuración de los componentes nativos

Cuadro 3.18: Sprint 5. Contacta con taller

- **US19 - Cambiar idioma.** Como usuario autenticado, independientemente de mi rol, quiero cambiar el idioma de mi aplicación.

Tareas
Realizar las instalaciones necesarias de las librerías de Ionic
Crear los archivos con las traducciones de los textos mostrados
Añadir las variables traducidas en todas las vistas de la aplicación

Cuadro 3.19: Sprint 5. Cambiar idioma

3.7.1 Diseño e implementación

- **US17 - Visualizar perfil**
  Para que el usuario pueda disponer de su información y la de sus promociones, se ha creado la vista **Perfil** en el *frontend* accesible desde el menú inferior de la aplicación y también desde el menú lateral. Se ha realizado una petición GET al servicio REST para obtener todos los datos.

- **US18 - Contactar con taller**
  En caso de que el usuario desee contactar con el taller mecánico para cualquier duda, se ha habilitado en el menú lateral la opción de **Contacta con nosotros**. Esta funcionalidad se desarrollo con `emailComposer` íntegramente en el *frontend*. Se debe verificar primero si el envío de correos electrónicos es compatible con el dispositivo en el que se lanza la aplicación.

```
1    this.emailComposer.isAvailable().then((available: boolean) =>{
2      if(available) {
3      } else {
4          logger.info('Envío de correos KO')
5      }
6    });
```

Listing 3.13: Verificación de envío de correos

Una vez realizada esta validación, se construyen los campos propios de un correo como el remitente o el asunto y se introduce el cuerpo que el usuario ha creado. Esta funcionalidad se ejecutará con Cordova, que lanzará la aplicación nativa de correo electrónico a la que se le enviarán todos los datos recabados, donde el usuario envía el email al taller mecánico.

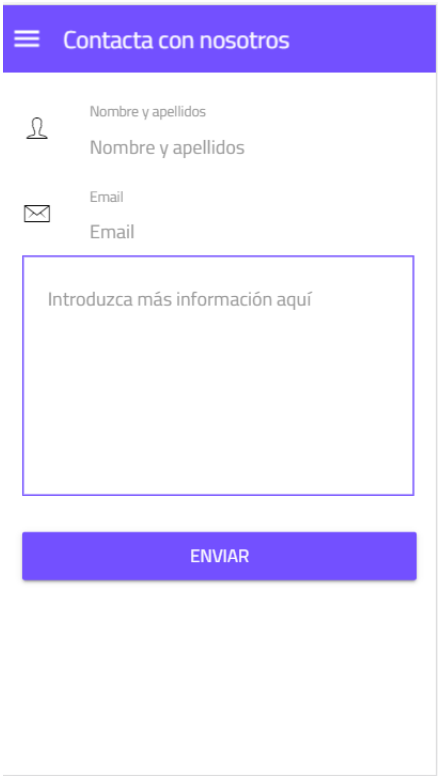


Figura 3.23: Formulario Contacta con Taller

- **US19 - Cambiar idioma**
  Se ha implementado la posibilidad de usar la aplicación tanto en español como en inglés. Al ser funcionalidades que ofrece Ionic, no ha sido necesario realizar un desarrollo en el servicio REST. Se ha usado la librería `ngx-translate` que asocia la traducción de un texto a una variable. Existe un fichero `.json` para cada idioma disponible en la aplicación, en el que hay una lista de variables con sus traducciones asociadas. Estos archivos se encuentran en la ruta **src/assets/i18n**.

  Es necesario especificar el lenguaje por defecto de la aplicación, el cual, para proporcionar una experiencia más cómoda al usuario, ha sido definido como el idioma del dispositivo en el que se ejecute la aplicación. Esta configuración se realiza en el archivo **app.component.ts**:

```
1    let language = translate.getBrowserLang();
2    this.translate.setDefaultLang(language);
3    this.translate.use(language);
```

Listing 3.14: Definición del idioma por defecto

```
{ //fichero es.json
    "SelecCoche": "Seleccione Coche",
    "SelecFecha": "Seleccione Fecha",
    "SelecHora":  "Seleccione Hora",
    "SelecTipo":  "Seleccione Tipo",
    "Horario1" :  "*Le recordamos nuestro horario de servicio",
    "Horario2" :  "Lunes a Viernes"
}
{    //fichero es.json
    "SelecCoche": "Choose Car",
    "SelecFecha": "Choose Date",
    "SelecHora":  "Choose Hour",
    "SelecTipo":  "Choose Type",
    "Horario1" :  "*Remember our business hours",
    "Horario2" :  "Monday - Friday",

}
```

Listing 3.15: Extracto de los ficheros de traducción

Por último y para acceder a esta configuración, se ha creado la vista de **Idiomas** accesible desde el menú lateral, en el que se puede cambiar el idioma de forma cómoda (ver figura 3.24).

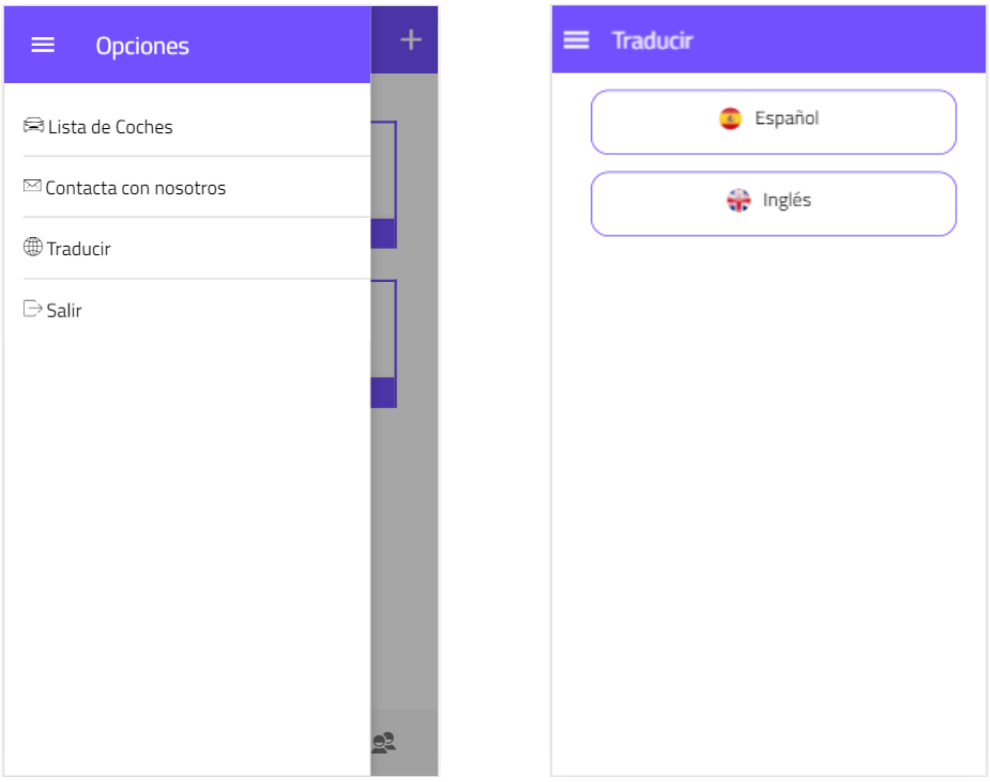


Figura 3.24: Menú traducción

3.8 Pruebas

Para asegurar la calidad del software de la aplicación se han realizado pruebas unitarias a medida que se iban desarrollando los diferentes *sprints*. Al comprobar el comportamiento de las diferentes clases existentes de forma aislada, se incrementa la confianza en un comportamiento correcto de las funcionalidades del código. Los errores cometidos son más fáciles de encontrar al probar los métodos existentes uno a uno a bajo nivel. Además de las pruebas unitarias, se han realizado pruebas de integración para evaluar el comportamiento general y la interacción entre las unidades del software. Estas validaciones fueron muy importantes en el **Sprint 2 (gestión de coches)** y en el **Sprint 3 (gestión de citas)** ya que la comunicación entre estas entidades es la base de toda la aplicación.

Capítulo 4
Conclusiones

En este capítulo se va a realizar una recapitulación del trabajo realizado así como una propuesta de líneas futura. Por último, la alumna aportará su valoración personal sobre el proyecto.

Una vez finalizado el proyecto, se considera que los objetivos marcados al inicio del mismo han sido logrados. Se ha implementado una aplicación móvil con las funcionalidades especificadas en el inicio de gestionar un taller de vehículos que al mismo tiempo ofrece a los dueños de los vehículos una herramienta para consultar los detalles de sus coches, concertar citas y visualizar sus facturas. Además, los usuarios pueden acceder a promociones consiguiendo bonificaciones en sus facturas. Para la gestión del taller, mediante un administrador, se pueden gestionar los usuarios dados de alta, así como realizar actualizaciones sobre sus citas. Se ha conseguido que el usuario se comunique con el taller de forma sencilla, lo que mejorará la relación entre el cliente y el establecimiento haciendo el servicio más colaborativo y, por lo tanto, atractivo.

Por otra parte, y teniendo en cuenta estos objetivos conseguidos, este proyecto ha implicado diseñar un sistema que cubre un nicho de mercado en la gestión de talleres que ninguna otra solución existente estaba cubriendo. A parte de tener una combinación de funcionalidades que abarca una gran parte de las necesidades existentes, su licencia libre, acerca la aplicación a establecimientos que de otra manera no podrían permitirse usar una infraestructura de este tipo.

4.1 Líneas futuras

Una vez el proyecto ha sido finalizado, se han detectado posibles mejoras a implementar en un futuro que podrían hacer la aplicación más atractiva y aportarle más valor:

- **Autenticación de redes sociales.** Actualmente, la autenticación clásica de correo electrónico y contraseña que se usa en esta aplicación empieza a ser limitada, por eso, sería

interesante incluir la autenticación con redes sociales para hacer el desarrollo más interesante. Al realizar la gestión de usuarios con Firebase, no sería necesario rediseñar el modelo existente aunque si habría que modificar los métodos implementados. La base para este desarrollo está realizada gracias al trabajo de la desarrolladora en este proyecto y únicamente habría que mejorar lo que ya existe.

- **Pago de las facturas.** Habilitar una pasarela de pago, para que, además de visualizar la factura de la reparación, ésta ya se pueda liquidar. Esta línea de desarrollo implicaría un esfuerzo importante a la hora de implementarlo ya que, al tratarse de operaciones bancarias, la seguridad es una prioridad. Sin embargo, el tiempo destinado a esta funcionalidad se vería recompensado con una gran revalorización del sistema al ofrecer un servicio que los usuarios valoran de forma muy positiva en otras aplicaciones.

4.2 Valoración personal

Finalmente, este proyecto ha supuesto un gran aporte de conocimiento para la alumna al usar tecnologías sobre las que no tenía conocimientos, sobre todo en la parte del *frontend*.

En concreto, Ionic ha sido la herramienta que no conocía y de la que más ha aprendido ya que ofrece una amplia variedad de funcionalidades para hacer la aplicación más atractiva. También los conocimientos adquiridos en Firebase son valorados muy positivamente.

Ha habido un afianzamiento de todo lo aprendido durante el grado, que ha supuesto una base sólida sobre la que construir todo el proyecto. En concreto, en el desarrollo con Java, la alumna ha podido poner en práctica los conocimientos adquiridos en asignaturas como Integración de Aplicaciones y mejorarlos ya que, siempre ha sido una tecnología que le ha causado conflictos y la necesidad de dedicar una cantidad de tiempo mayor a la programada en las asignaturas.

Bibliografía

[1] "Connection soft service." [En línea]. Disponible en: https://www.css.es/italler.html

[2] "Opiniones autingo." [En línea]. Disponible en: https://es.trustpilot.com/review/www.autingo.es

[3] "Opiniones de autingo." [En línea]. Disponible en: https://es.trustpilot.com/review/www.autingo.es?page=7

[4] "Scrum." [En línea]. Disponible en: https://proyectosagiles.org/que-es-scrum/

[5] "Productividad scrum." [En línea]. Disponible en: https://www2.deloitte.com/es/es/pages/technology/articles/que-es-scrum.html

[6] J. S. y J.J Sutherland, *Scrum: El revolucionario método para trabajar el doble en la mitad de tiempo.* Ariel, 2008.

[7] "Trello." [En línea]. Disponible en: https://trello.com/es

[8] "Java." [En línea]. Disponible en: https://java.com/es/download/faq/whatis_java.xml

[9] "Hibernate." [En línea]. Disponible en: https://hibernate.org/

[10] "Spring." [En línea]. Disponible en: https://spring.io/why-spring

[11] "Html." [En línea]. Disponible en: https://desarrolloweb.com/home/html

[12] C. Griffith, *Mobile APP Development with Ionic.* O'Reilly, 2017.

[13] "Typescript." [En línea]. Disponible en: https://es.wikipedia.org/wiki/TypeScript

[14] "Javascript." [En línea]. Disponible en: https://es.wikipedia.org/wiki/JavaScript

[15] "Eclipse." [En línea]. Disponible en: https://www.eclipse.org

[16] "pgadmin." [En línea]. Disponible en: https://www.pgadmin.org

[17] "Visual studio code." [En línea]. Disponible en: https://code.visualstudio.com/

[18] "Git." [En línea]. Disponible en: https://git-scm.com/

[19] "Diagrama de gantt." [En línea]. Disponible en: https://es.wikipedia.org/wiki/Diagrama_de_Gantt

[20] "Json web token." [En línea]. Disponible en: https://jwt.io/

[21] "Navctrl." [En línea]. Disponible en: https://ionicframework.com/docs/v3/api/navigation/NavController/