Aplicación para el seguimiento de repartos en una empresa de transportes para administradores, repartidores y clientes

PARTE 2

Tutor: Abraham Rodríguez Rodríguez

Autor: María del Rosario Guerra Suárez

Fecha de entrega: julio de 2017

Índice

1. Estado actual y objetivos iniciales

1.1. Introducción

Este proyecto nace a partir de la idea de mejorar el sistema actual de mensajería y adaptarlo a las nuevas tecnologías disponibles, de forma que suponga una mejora en la productividad, usabilidad y simplicidad. Al tratarse de un proyecto de gran envergadura, su desarrollo comenzó de forma conjunta con otro alumno, Miguel González.

Se identificaron tres agentes principales en el proceso de entrega de un paquete, el administrador del almacén, el repartidor y el cliente (receptor del paquete). El trabajo de los dos primeros podía agilizarse y simplificarse, mientras que el tercer agente requería una ampliación de la información que las compañías de repartos suelen proporcionar. Para lograr la movilidad necesaria en esta situación, se desarrollan tres aplicaciones móviles para cada uno de los agentes.

La primera parte de este proyecto se centrará en la aplicación del administrador del almacén, cuyo trabajo consiste en la capacidad de visualizar la información completa de un pedido entrante y asignarlo a cualquier repartidor disponible, obtener en tiempo real la localización de los repartidores y controlar y asignar vehículos de entre los disponibles a repartidores, entre otras tareas.

1.2. Descripción de objetivos

Identificados los requisitos propuestos de la aplicación del administrador del almacén, además de los que el tutor ha creído convenientes añadir, se pretende desarrollar un prototipo de aplicación móvil que ejemplifique el trabajo de desarrollo en un entorno real. Dicho prototipo puede ampliarse hasta obtener una versión final con la debida financiación por parte de una empresa de transportes interesada.

Si bien se trata de un prototipo de aplicación, y por ello presenta algunas limitaciones, se ha tratado de realizar un trabajo muy cercano a la realidad. Debido al carácter educativo del TFT, los principales objetivos propuestos por el estudiante durante el desarrollo del proyecto son los siguientes:

- Aprendizaje de una tecnología de reciente desarrollo como es Ionic 3 y Angular 4.
- Mejora en los conocimientos de diseño y desarrollo web (HTML5 y CSS3) obtenidos hasta ahora.
- Adquirir habilidad en el uso del lenguaje de programación Typescript.
- Familiarizarse con el uso de librerías y API's móviles para implementar y aprovechar las funcionalidades y servicios de un dispositivo móvil.
- Aprender a desenvolverse con un servidor de almacenamiento remoto como Firebase.

Como se puede observar, los objetivos están muy centrados en adquirir los conocimientos necesarios para el desarrollo de aplicaciones en Ionic 3, un *framework* sustentado sobre Angular 4,

TypesScript 2.2, HTML5 y CSS3. La decisión de usar un entorno de desarrollo que apenas tiene unos meses de vida se debe al deseo de trabajar desde un punto muy temprano en un *framework* que probablemente acapare el desarrollo de aplicaciones web en un futuro cercano, con las ventajas que ello proporciona a nivel laboral. Es notable la falta de documentación con respecto a desarrollar en otras plataformas como Android (que se sustenta en Java) o iOS, pero se compensa con una interfaz sencilla y la posibilidad de construir aplicaciones para ambos dispositivos anteriormente nombrados sin la necesidad de aprender a trabajar en cada uno por separado.

1.3. Metodología

Se ha utilizado una metodología convencional en cascada para la realización de este trabajo. Se trata de una metodología que permite ordenar de manera rigurosa las fases del proceso del desarrollo del software. Cada etapa posee ciertos objetivos que se deben cumplir siguiendo un conjunto de actividades específicas para cada una. En la *Imagen I* se puede observar las distintas fases del modelo en cascada.

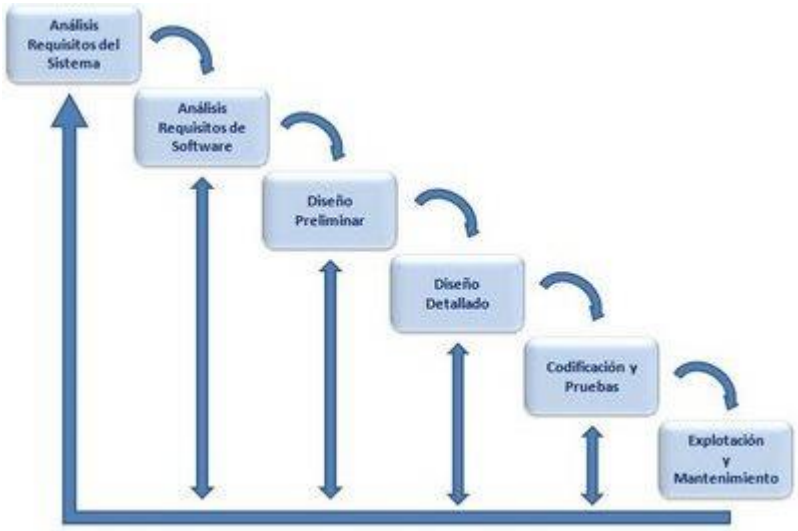

Imagen I: Etapas de la metodología en cascada

En el TFT01 se especificaron las siguientes tareas a realizar:

- Asignación de pedidos a repartidores.

- Recogida de datos del cliente o la empresa solicitante.

- Visualización de repartidores y vehículos libres y ocupados.

- Seguimiento de pedidos.

- Envío de notificaciones a clientes y repartidores.

Aunque la mayoría de estas tareas se han implantado e incluso se ha ampliado a algunas más que no aparecen, hay una de ellas que se ha decidido modificar. La recogida de datos del cliente o empresa solicitante ya no es trabajo del administrador de la empresa. El administrador recibe los paquetes ya creados y sólo debe encargarse de revisarlos y asignarlos a los repartidores.

2. Justificación de las competencias específicas cubiertas

Las competencias que cubre este proyecto comienzan con el aprendizaje de una nueva área de conocimiento y el desarrollo de la misma, además del despliegue del trabajo realizado. Todo ello sin descuidar la importancia del trabajo en equipo o el conocimiento de la legislación vigente. Pueden destacarse las siguientes competencias:

G1: *Poseer y comprender conocimientos en un área de estudio (Ingeniería Informática) que parte de la base de la educación secundaria general, y se suele encontrar a un nivel que, si bien se apoya en libros de texto avanzados, incluye también algunos aspectos que implican conocimientos procedentes de la vanguardia de su campo de estudio.*

En este apartado se puede destacar el aprendizaje relativo a las nuevas tecnologías que forman el proyecto, como Ionic 3 o el refuerzo de los conocimientos en desarrollo web que se adquirieron a lo largo del periodo universitario.

G2: *Aplicar sus conocimientos a su trabajo o vocación de una forma profesional y posean las competencias que suelen demostrarse por medio de la elaboración y defensa de argumentos y la resolución de problemas dentro de su área de estudio.*

El proyecto desarrollado es una buena prueba de la aplicación de los conocimientos adquiridos y la resolución de un problema planteado como es la mejora de la productividad en el sector de transportes, teniendo en cuenta especialmente la profesionalidad con la que se ha trabajado para obtener unas aplicaciones lo más cercanas posibles al mundo real.

N2: *Cooperar con otras personas y organizaciones en la realización eficaz de funciones y tareas propias de su perfil profesional, desarrollando una actitud reflexiva sobre sus propias competencias y conocimientos profesionales y una actitud comprensiva y empática hacia las competencias y conocimientos de otros profesionales.*

Al tratarse de un proyecto que se ha desarrollado en colaboración con otro alumno, la cooperación ha sido clave para resolver los problemas encontrados. Las tres aplicaciones que conforman la totalidad del proyecto deben su éxito al trabajo en equipo.

T7: *Capacidad para conocer, comprender y aplicar la legislación necesaria durante el desarrollo de la profesión de Ingeniero Técnico en Informática y manejar especificaciones, reglamentos y normas de obligado cumplimiento.*

Puesto que las aplicaciones pretenden mejorar el proceso de gestión y reparto de una empresa de transporte, era inevitable tratar con datos personales, tanto de los trabajadores de la empresa como los receptores de paquetes. Por ello, se ha elaborado el Documento de Seguridad de la LOPD (Ley Orgánica de Protección de Datos), que se adjunta a esta memoria.

TI01: *Capacidad para comprender el entorno de una organización y sus necesidades en el ámbito de las tecnologías de la información y las comunicaciones.*

El primer reto que se ha debido superar a la hora de realizar este proyecto es justamente esta competencia. Hasta que no se identificaron las necesidades de una empresa de transportes no se pudo desarrollar un proyecto que cumpliera dichas necesidades y garantizase un manejo fiable y seguro.

TI02: *Capacidad para seleccionar, diseñar, desplegar, integrar, evaluar, construir, gestionar, explotar y mantener las tecnologías de hardware, software y redes, dentro de los parámetros de coste y calidad adecuados.*

La sincronización entre las aplicaciones que forman el proyecto es una necesidad que no puede evitarse, pero hay una pieza clave en el desarrollo y se trata del servidor de almacenamiento empleado. Dicho servidor se emplea como gestor de autenticación de los usuarios, gestor de la base de datos y como espacio de almacenamiento, por lo que la correcta configuración del mismo (además de la correcta ejecución de las operaciones de lectura y escritura realizadas desde las aplicaciones) son vitales para un funcionamiento idóneo.

TI06: *Capacidad de concebir sistemas, aplicaciones y servicios basados en tecnologías de red, incluyendo Internet, web, comercio electrónico, multimedia, servicios interactivos y computación móvil.*

El desarrollo del proyecto ha permitido alcanzar esta competencia con facilidad, todo ello gracias al planteamiento móvil de las aplicaciones, pero también al desarrollo basado en tecnologías web que aporta Ionic 3.

TFG01: *Ejercicio original a realizar individualmente y presentar y defender ante un tribunal universitario, consistente en un proyecto en el ámbito de las tecnologías específicas de la Ingeniería en Informática de naturaleza profesional en el que se sinteticen e integren las competencias adquiridas en las enseñanzas.*

Todos los conocimientos adquiridos a lo largo del periodo universitario se condensan en un proyecto con grandes proyecciones a un cercano futuro profesional, proyecto que se encuentra detallado en una memoria final dividida en dos partes.

3. Aportaciones al entorno socio-económico, técnico o científico

Las aportaciones que este proyecto puede proporcionar están centradas en el aprendizaje y habilidad de manejo del alumno ante una tecnología tan reciente y prometedora como es Ionic. También, permite extender y dar a conocer este framework tan multidimensional. Es sabido que no se dispone de tanta documentación como ocurre con otras tecnologías y que, además, hay cambios en ciertos aspectos de implementación con frecuencia debido a sus actualizaciones, lo que supone un desafío para el alumno al requerir una renovación constante de los conocimientos adquiridos.

Este proyecto no sólo aporta conocimientos de desarrollo web, sino que además supone una importante mejora en la capacidad de trabajar en equipo, gracias a tratarse de un proyecto conjunto. Esta capacidad para trabajar en un entorno colaborativo tiene una vital consideración en el ámbito laboral, lo que otorga a esta aportación una importancia inigualable.

No obstante, pese a considerarse el proyecto como un prototipo y ser perfectamente funcional, carece de un equipo de trabajo lo suficientemente grande como para garantizar la mejora continua de la aplicación. Así mismo, sería conveniente disponer de un servidor propio sobre el que almacenar un flujo de datos constante. El proyecto podría continuar mejorándose hasta obtener una versión final con muchas opciones de ser empleado en una empresa real de reparto.

4. Desarrollo

El proyecto está formado por un conjunto de tres aplicaciones, pero en esta memoria solo se mostrará el desarrollo de la primera parte de dicho proyecto, fundamentada en la aplicación diseñada para el administrador del almacén.

4.1.Análisis de aplicaciones similares

Antes de comenzar con el análisis de los requisitos del proyecto, es importante valorar las soluciones software existentes de forma que el desarrollo del prototipo pueda aportar algún valor tecnológico de interés y diferenciarse con funcionalidades nuevas que resulten de utilidad para la tarea concreta que se pretende desarrollar.

Los sistemas empleados en el campo de la distribución de paquetería se basan en los SGA, cuyas siglas se corresponden a “Software de Gestión de Almacén”. Estos sistemas se encargan de facilitar la gestión de almacenes y automatización de control de mercancía, además del registro de entradas y salidas de mercancía.

En primer lugar está “SIGA”, un sistema SGA con tecnología inalámbrica desarrollado por *Inisoft* que gracias al empleo de dispositivos móviles permite disminuir el tiempo por operación, mantener un inventario organizado y perfectamente adaptable a las necesidades de la empresa gracias a un periodo de adaptación e instalación en la misma empresa, incluyendo la formación de

los operarios que emplearán el sistema. La gran cantidad de acciones que implementa podría considerarse un defecto debido a la dificultad de uso que repercute en el sistema.

La segunda aplicación analizada es "LOGISGEST SGA", creada por *Tekktia* y basa su funcionamiento completamente en dispositivos móviles. Cuenta con una interfaz sencilla que permitirá manejar cualquier aspecto de logística desde la palma de la mano, aumentando el rendimiento, mejorando los plazos de entrega y permitiendo mantener un control de la trazabilidad de los productos. Entre sus desventajas se encuentra su diseño poco atractivo y complejo de emplear en dispositivos móviles.

Finalmente, el tercer sistema es "OnERP", un software online desarrollado por *Web Evolutive Business Software S.L* que, basándose en el uso de módulos es capaz de cubrir las necesidades de cualquier empresa que quiera mejorar sus sistemas en los campos de gestión de recursos humanos, ventas, almacenamiento, contabilidad, gestión de proyectos, producción, calidad o seguridad entre otros. Se encuentra el módulo de almacén, un programa para gestión de almacenes que posee control de existencias y trazabilidad completa, así como control de expediciones con su correspondiente conexión con dispositivos móviles. Por tratarse de una solución "online", es posible acceder desde cualquier dispositivo, ya sea un ordenador que se encuentre en el almacén o un "Smartphone".

Los sistemas anteriormente descritos cuentan con un equipo que trabaja en ellos, actualizando y proporcionando mantenimiento a las empresas que los adquiera, por lo que la calidad de dichos sistemas está garantizada, así como su fiabilidad. Para que el proyecto suponga una diferenciación en un campo como la logística es importante fijarse en los grandes aciertos de otras compañías de "software" a la hora de diseñar sus productos. Es por ello que, además de requerir funcionalidades de control y asignación de paquetería a repartidores, un sistema dedicado a un gestor de almacén requerirá una interfaz sencilla, pero con las operaciones debidamente establecidas y en perfecta sincronización, como requiere el sistema de geolocalización de repartidores. Para cumplir estas características se desarrollará la aplicación teniendo en mente dispositivos móviles, pero pudiendo ser empleada en equipos de escritorio simultáneamente.

4.2. Análisis de requisitos

El primer paso del desarrollo del proyecto consiste en realizar un análisis exhaustivo de las funcionalidades que son necesarias para la realización de un correcto trabajo por parte del administrador del almacén. Los requisitos funcionales son:

1. Asignación de pedidos a repartidores.
2. Asignación de vehículos a repartidores.
3. Posibilidad de reportar un pedido cuyos datos sean incorrectos.
4. Visualización en tiempo real de la posición de los repartidores.
5. Visualización en tiempo real de los puntos de entrega de paquetes.
6. Posibilidad de comunicarse con los repartidores a través de la aplicación.
7. Visualización de un historial de pedidos entregados.
8. Restricción de acceso a usuarios no identificados.

Los requisitos no funcionales son aquellos que especifican características de funcionamiento. Entre ellos están:

1. **Rendimiento:** La aplicación requiere una velocidad de ejecución de operaciones lo suficientemente alta como para no perjudicar el flujo de trabajo del usuario que la emplee. El código empleado deberá ser ligero y rápido para funcionar con normalidad en la mayoría de dispositivos portátiles del mercado.
2. **Estabilidad:** Es preciso que la aplicación no presente errores durante su uso, puesto que podría causar problemas en la gestión de mercancías y repartidores o vehículos.
3. **Portabilidad:** Un requisito importante es que la aplicación pueda ser empleada en un dispositivo portátil como podría ser un equipo pequeño o un "Smartphone".
4. **Usabilidad:** La aplicación presentará una gran sencillez de uso y facilidad de acceso a cualquier punto de la misma en pocos pasos, lo que garantizará una aplicación fluida e intuitiva.

4.2.1. Definición de tareas

Una vez definidas las funcionalidades, es necesario establecer una serie de requisitos para cada una, lo que desembocará en un listado de tareas que serán implementadas en la aplicación.

1. Asignación de pedidos a repartidores.

El administrador deberá ser capaz de seleccionar un pedido y asignarlo a un repartidor a su elección. Para cumplir con esta funcionalidad, se han definido las siguientes tareas:

- Mostrar en la pantalla principal los pedidos que se encuentran en el almacén sin asignar, con información breve sobre cada uno.
- Crear una página para mostrar en detalle cada pedido y desde donde se podrá asignar el pedido seleccionado a un repartidor de entre una lista (ordenada en función del número de pedidos que tuviera asignado cada uno).
- Implementar la opción de desasignar un pedido.
- Posibilidad de asignar múltiples paquetes a un repartidor.

2. Asignación de vehículos a repartidores.

El administrador deberá ser capaz de seleccionar un repartidor y asignarle un vehículo para las entregas de pedidos. Se cumplirá esta funcionalidad a raíz de completar las siguientes tareas:

- Crear una página para mostrar en detalle la información de cada repartidor y desde donde se asignará un vehículo de entre una lista de vehículos disponibles (no asignados a otro repartidor).
- Implementar la opción de desasignar un vehículo (tanto desde la vista del repartidor como desde la vista general de repartidores).

3. *Posibilidad de reportar un pedido cuyos datos sean incorrectos.*

En caso de encontrarse un pedido cuyos datos sean incorrectos, el administrador tendrá la capacidad de marcarlos como erróneos y añadir una observación al respecto. Las tareas previstas son:

- En el panel de visualización de pedidos, cada pedido contará con un botón que servirá para introducir la observación y marcar el pedido como erróneo.
- Los pedidos erróneos se mostrarán en otra sección de la página principal.

4. *Visualización en tiempo real de la posición de los repartidores.*

El administrador requiere conocer la constante posición de los repartidores, de forma que podrá priorizar la asignación de pedidos a los repartidores que se encuentren más cerca del almacén. Así mismo, podrá observar si los repartidores incumplen con sus tareas. Las tareas necesarias son:

- Implementación del módulo del mapa y creación de la página para mostrarlo.
- El mapa deberá ser accesible desde la vista general de los repartidores o desde el menú lateral.
- Creación de marcadores que se actualicen con los cambios de posición del repartidor al que pertenecen.
- Los marcadores deberán mostrar el nombre y estado del repartidor al que se asocian, de forma que, con un sencillo código de dos colores, se podrá conocer si el repartidor tiene entregas pendientes en su vehículo o regresa al almacén para recoger pedidos. Esta distinción se realiza para evitar asignar nuevos pedidos a un repartidor ocupado, pero si a uno de vuelta al almacén.

5. *Visualización en tiempo real de los puntos de entrega de paquetes.*

El administrador dispondrá de la posición de entrega de todos los paquetes, excepto los entregados. Esta funcionalidad permitirá al administrador optimizar la asignación de pedidos a los repartidores, de forma que evite asignar pedidos con mucha distancia entre sí al mismo repartidor, además de observar en tiempo real las variaciones de estado de los pedidos.

- En la pantalla principal de la aplicación se mostrará el mapa con cada pedido.
- El marcador asignado a cada pedido cambiará de color en función del estado del pedido, indicando hasta cuatro colores distintos.

6. *Posibilidad de comunicarse con los repartidores a través de la aplicación.*

El administrador tendrá la posibilidad de comunicarse con los repartidores en privado en cualquier momento sin salir de la aplicación. De esta forma podrá informar personalmente a los repartidores de los paquetes que le han sido asignados, proporcionar alguna observación sobre algún pedido en concreto o simplemente comprobar si algún repartidor está libre o no para recibir más paquetes.

- En la pantalla individual de cada repartidor se mostrará un botón con el que acceder a una página nueva desde la que intercambiar mensajes con el repartidor deseado.
- La conversación con cada repartidor será accesible desde una página dedicada a la mensajería. Dicha página de mensajería será accesible desde el menú lateral.

7. Visualización de un historial de pedidos entregados

El trabajo diario del administrador requiere también la posibilidad de consultar el historial de pedidos entregados, en caso de existir una reclamación por parte del cliente, una petición de consulta por parte de la empresa remitente o para la consulta, estadística y contabilidad posteriores. Las tareas a realizar son:

- Creación de una nueva página para mostrar el histórico.
- Introducción de un sistema de ordenación y filtrado por múltiples campos.

8. Restricción de acceso a usuarios no identificados.

Será necesario que el usuario, que en este caso será solamente el administrador del almacén, se identifique para poder acceder a la aplicación. De esta manera se controla el acceso de usuarios no autorizados.

- Sistema de autenticación al inicio de la aplicación basado en correo electrónico y clave.
- Sistema de cierre de sesión.

4.3. Diagrama de casos de uso

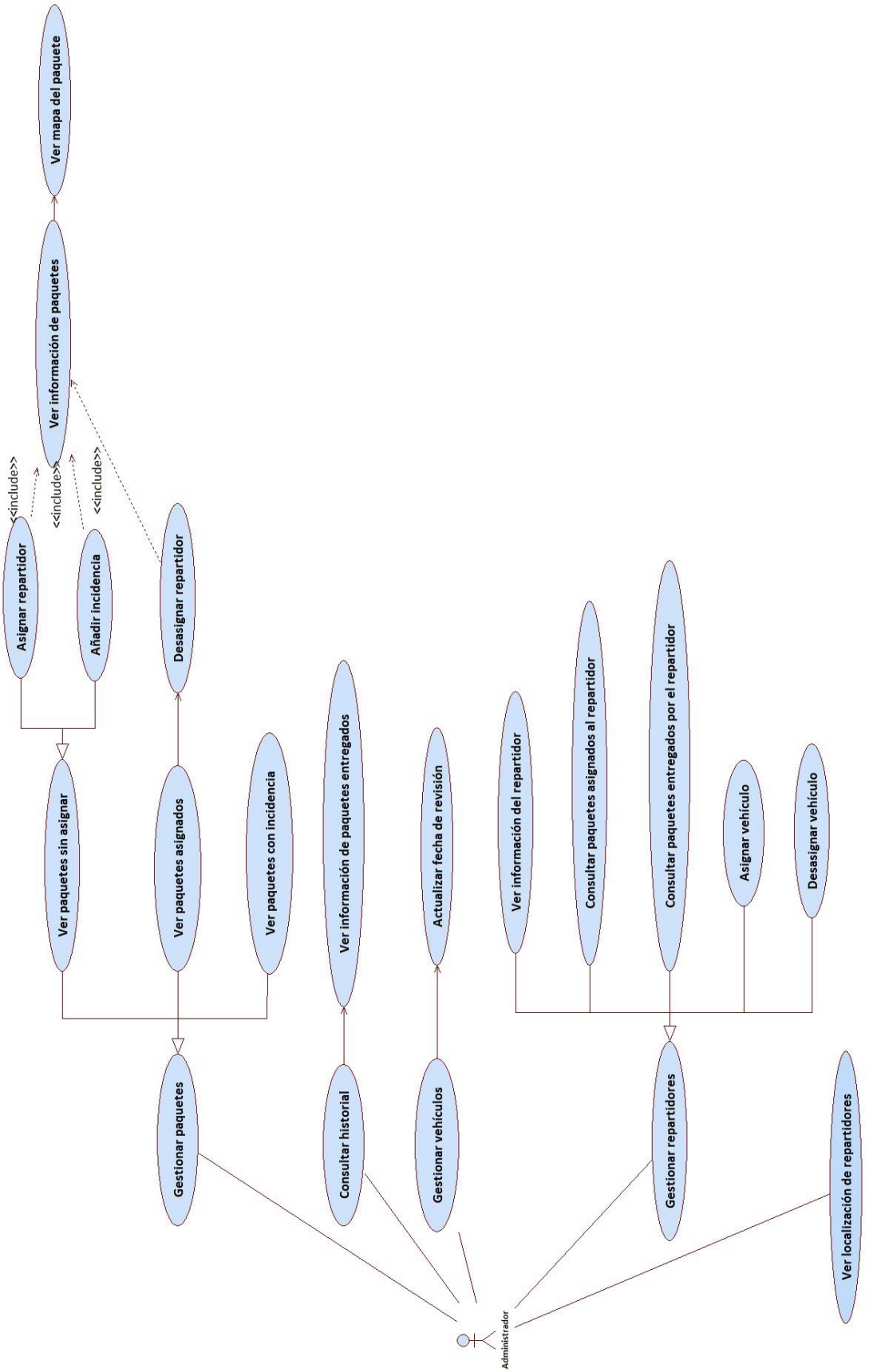

Imagen II: Diagrama de casos de uso

A continuación, se detalla cada uno de los casos de uso que se reflejan en la *Imagen II*.

CU-001	Gestionar paquetes
Versión	1.0
Descripción	Es la parte de la aplicación que se encarga de los paquetes.
Secuencia normal	1. Iniciar sesión en la aplicación.
Frecuencia	Alta
Importancia	Alta

CU-002	Ver paquetes sin asignar
Versión	1.0
Descripción	El administrador puede ver todos los paquetes que están en el almacén y que aún no han sido asignados a un repartidor
Secuencia normal	1. Iniciar sesión en la aplicación. 2. Seleccionar segmento “Sin asignar” en la página principal o portada.
Frecuencia	Alta
Importancia	Alta

CU-003	Ver paquetes asignados
Versión	1.0
Descripción	El administrador puede ver todos los paquetes que están asignados a los repartidores
Secuencia normal	1. Iniciar sesión en la aplicación. 2. Seleccionar segmento “Asignados” en la página principal o portada.
Frecuencia	Alta
Importancia	Alta

CU-004	Ver paquetes con incidencia
Versión	1.0
Descripción	El administrador puede tener la lista de paquetes que ha marcado con alguna incidencia.
Secuencia normal	1. Iniciar sesión en la aplicación. 2. Seleccionar segmento “Incidencias” en la página principal o portada.
Frecuencia	Baja
Importancia	Alta

CU-005	Ver información de paquetes
Versión	1.0
Descripción	El administrador tiene la posibilidad de ver toda la información referente a un paquete específico.
Secuencia normal	1. Iniciar sesión en la aplicación. 2. Seleccionar un paquete del segmento “Sin Asignar” o “Asignados”.
Frecuencia	Alta
Importancia	Alta

CU-006	Asignar repartidor
Versión	1.0
Descripción	El paquete seleccionado es asignado a un repartidor para su posterior envío.
Secuencia normal	1. Iniciar sesión en la aplicación. 2. Seleccionar un paquete del segmento “Sin asignar”. 3. Seleccionar un repartidor del slider. 4. Confirmar.
Frecuencia	Alta
Importancia	Alta

CU-007	Desasignar repartidor
Versión	1.0
Descripción	El paquete seleccionado es desasignado a un repartidor.
Secuencia normal	1. Iniciar sesión en la aplicación. 2. Seleccionar un paquete del segmento “Asignados”. 3. Pulsar el botón “Desasignar repartidor”.
Frecuencia	Baja
Importancia	Alta

CU-008	Añadir incidencia
Versión	1.0
Descripción	Se detalla una incidencia en un paquete y se marca como erróneo.
Secuencia normal	1. Iniciar sesión en la aplicación. 2. Seleccionar un paquete del segmento “Sin asignar”. 3. Pulsar el botón con el icono de advertencia. 4. Escribir la incidencia. 5. Confirmar.
Frecuencia	Alta
Importancia	Alta

CU-009	Ver mapa del paquete
Versión	1.0
Descripción	Se muestra en un mapa la dirección donde debe ser entregado el paquete.
Secuencia normal	1. Iniciar sesión en la aplicación. 2. Seleccionar un paquete del segmento “Sin asignar” o “Asignados”. 3. Pulsar el botón con el icono de navegación.
Frecuencia	Alta
Importancia	Alta

CU-010	Consultar historial
Versión	1.0
Descripción	El administrador puede ver una lista de paquetes que se han entregado, ordenada por la fecha de entrega.
Secuencia normal	1. Iniciar sesión en la aplicación. 2. Seleccionar “Historial” en el menú lateral.
Frecuencia	Alta
Importancia	Alta

CU-011	Ver información de paquetes entregados
Versión	1.0
Descripción	Se muestra toda la información de un paquete ya entregado.
Secuencia normal	1. Iniciar sesión en la aplicación. 2. Seleccionar “Historial” en el menú lateral. 3. Seleccionar un paquete de la lista.
Frecuencia	Alta
Importancia	Alta

CU-012	Gestionar vehículos
Versión	1.0
Descripción	Es la parte de la aplicación centrada en los vehículos de empresa.
Secuencia normal	1. Iniciar sesión en la aplicación. 2. Seleccionar “Vehículos” en el menú lateral.
Frecuencia	Baja
Importancia	Media

CU-013	Actualizar fecha de revisión
Versión	1.0
Descripción	Se cambia la fecha de la última revisión pasada por un vehículo.
Secuencia normal	1. Iniciar sesión en la aplicación. 2. Seleccionar “Vehículos” en el menú lateral. 3. Pulsar el botón con el icono de una herramienta. 4. Confirmar.
Frecuencia	Baja
Importancia	Media

CU-014	Gestionar repartidores
Versión	1.0
Descripción	Es la parte de la aplicación encargada de la gestión de todos los repartidores.
Secuencia normal	1. Iniciar sesión en la aplicación. 2. Seleccionar “Vehículos” en el menú lateral. 3. Pulsar el botón con el icono de una herramienta. 4. Confirmar.
Frecuencia	Alta
Importancia	Alta

CU-015	Ver información del repartidor
Versión	1.0
Descripción	El administrador puede consultar la información detallada de cada repartidor.
Secuencia normal	1. Iniciar sesión en la aplicación. 2. Seleccionar “Repartidores” en el menú lateral. 3. Seleccionar uno de los repartidores de la lista.
Frecuencia	Alta
Importancia	Alta

CU-016	Consultar paquetes asignados al repartidor
Versión	1.0
Descripción	Consultar los paquetes asignados a ese repartidor.
Secuencia normal	1. Iniciar sesión en la aplicación. 2. Seleccionar “Repartidores” en el menú lateral. 3. Seleccionar uno de los repartidores de la lista. 4. Seleccionar pestaña inferior “Asignados”.
Frecuencia	Media
Importancia	Media

CU-017	Consultar paquetes entregados por el repartidor
Versión	1.0
Descripción	Consultar los paquetes que han sido entregados por ese repartidor.
Secuencia normal	1. Iniciar sesión en la aplicación. 2. Seleccionar “Repartidores” en el menú lateral. 3. Seleccionar uno de los repartidores de la lista. 4. Seleccionar pestaña inferior “Entregados”.
Frecuencia	Media
Importancia	Media

CU-018	Asignar vehículo
Versión	1.0
Descripción	Asignación de un vehículo a un repartidor.
Secuencia normal	1. Iniciar sesión en la aplicación. 2. Seleccionar “Repartidores” en el menú lateral. 3. Seleccionar uno de los repartidores de la lista. 4. Pulsar el botón con el icono de un coche. 5. Confirmar.
Frecuencia	Alta
Importancia	Alta

CU-019	Desasignar vehículo
Versión	1.0
Descripción	Desasignar un vehículo a un repartidor.
Secuencia normal	1. Iniciar sesión en la aplicación. 2. Seleccionar “Repartidores” en el menú lateral. 3. Seleccionar uno de los repartidores de la lista. 4. Pulsar el botón “Desasignar vehículo”. 5. Confirmar.
Frecuencia	Alta
Importancia	Alta

CU-020	Ver localización de repartidores
Versión	1.0
Descripción	El administrador puede consultar la localización de los repartidores que están en ruta de reparto.
Secuencia normal	1. Iniciar sesión en la aplicación. 2. Seleccionar “Localización de repartidores”.
Frecuencia	Alta
Importancia	Alta

4.4. Mockups

Una vez creados los casos de uso, era el momento de diseñar unas muestras de la interfaz sobre la que tener una base para el prototipo. En primer lugar, y analizando las funcionalidades requeridas, se optó por un diseño que mostrase los pedidos entrantes en la ventana principal, para disponer de la mayor cantidad de información relevante lo más cerca posible del inicio de la aplicación. El resto de páginas serían accesibles a través de un menú lateral. Reducir el número de transiciones entre páginas es muy importante para obtener una aplicación que se pueda operar con rapidez, razón por la que la página principal concentra la mayor parte de la zona de trabajo del gestor del almacén, siendo incluso posible controlar la mayor parte de la aplicación con una sola transición entre páginas. Como se puede ver en los diseños a continuación, el objetivo es una aplicación que presente una interfaz capaz de mejorar la productividad de quién la emplee.

En la *Imagen III* se puede ver la página de acceso a la aplicación, donde el adminsitrador del almacén deberá introducir su dirección de correo electrónico y su clave.

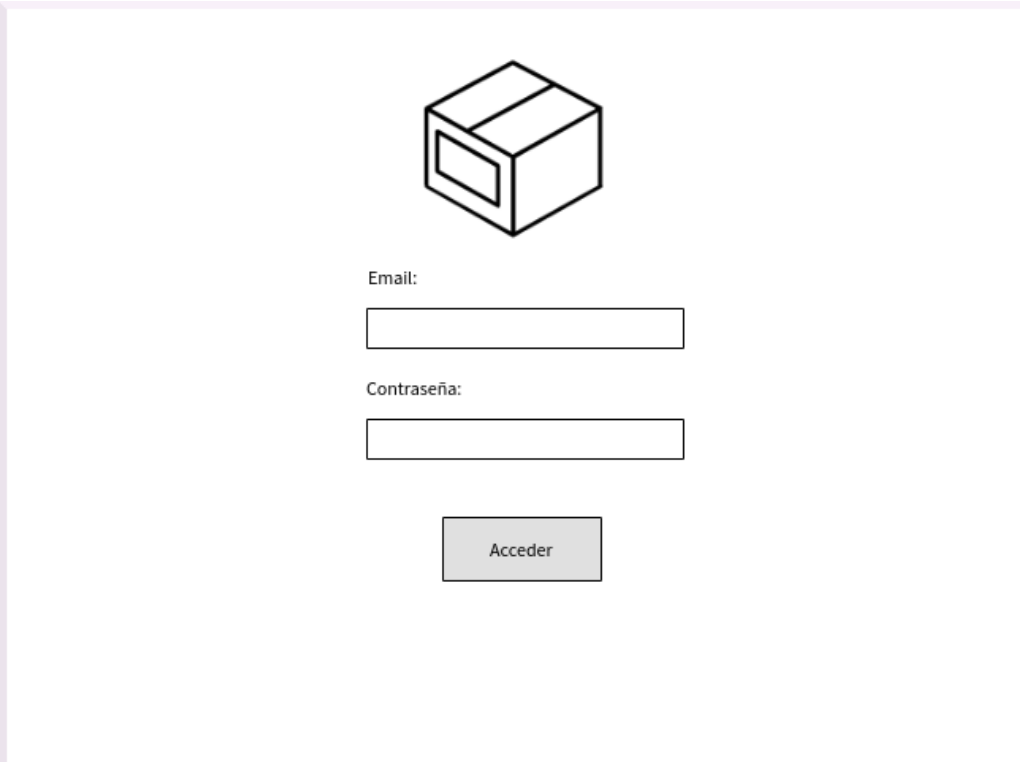

Imagen III: Página de autenticación

Una vez iniciada la sesión, aparecerá la página principal o portada, que corresponde a la *Imagen IV*, donde se gestionan los paquetes tanto sin asignar, como asignados y con incidencias. El diseño de los tres segmentos sería el mismo.

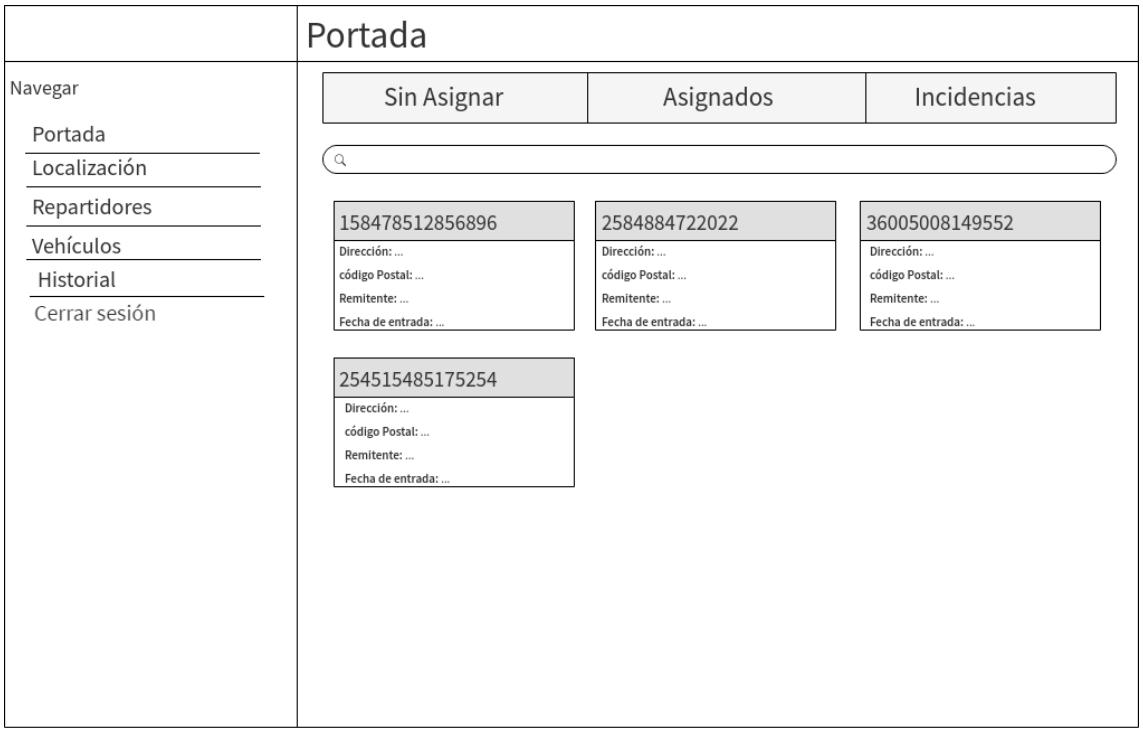

Imagen IV: Página principal

En la *Imagen V* se observa la página de información de un paquete, a la cual se puede acceder seleccionando un paquete tanto del segmento "Sin Asignar" como del "Asignados". Se entiende que los paquetes con incidencias son paquetes erróneos a los que no le interesa acceder al gestor del almacén. El responsable de crear los paquetes en la aplicación será el encargado de modificarlos y habilitarlos de nuevo.

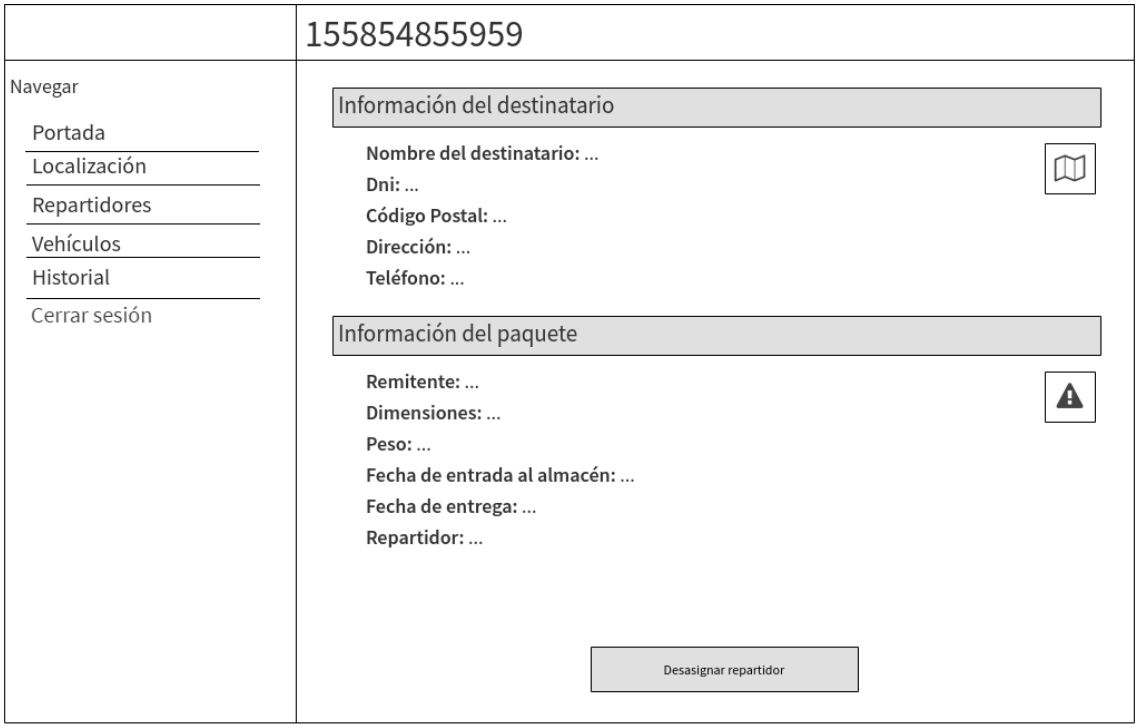

Imagen V: Página de información de paquete

En este caso, el paquete ya está asignado a un repartidor, por lo que aparece la opción de desasignarlo. Si no estuviera asignado a ninguno, aparecería un slider con todos los repartidores, ordenado de forma ascendente por el número de paquetes ya asignados a éstos. Es decir, los repartidores con menor número de paquetes asignados serán los primeros.

El menú lateral estático permite al usuario desplazarse por las diferentes páginas de la aplicación de forma sencilla y rápida. Accediendo a la opción "Localización" de este menú, el usuario puede ver la ubicación en tiempo real de todos los repartidores que están en ruta en ese momento, tal y como se muestra en la *Imagen VI*.

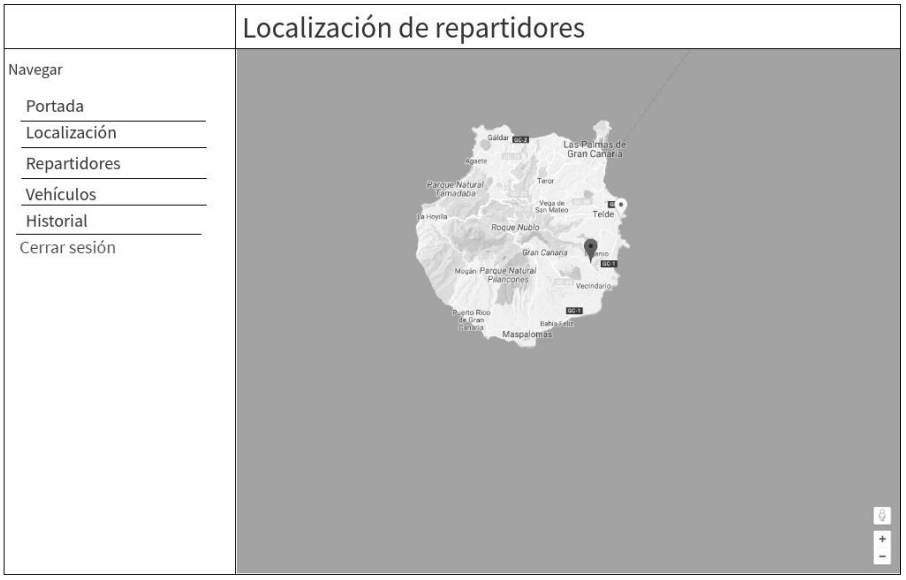

Imagen VI: Página de localización de repartidores

La opción "Repartidores" del menú lateral conduce a la página encargada de gestionar los repartidores (*Imagen VII*). En ella se muestra una lista de todos ellos y su información relevante.

Los iconos a la derecha de cada tarjeta corresponden a la información del vehículo. Si tienen un vehículo asignado aparecerá una "X" y si no, aparecerá el icono de un coche.

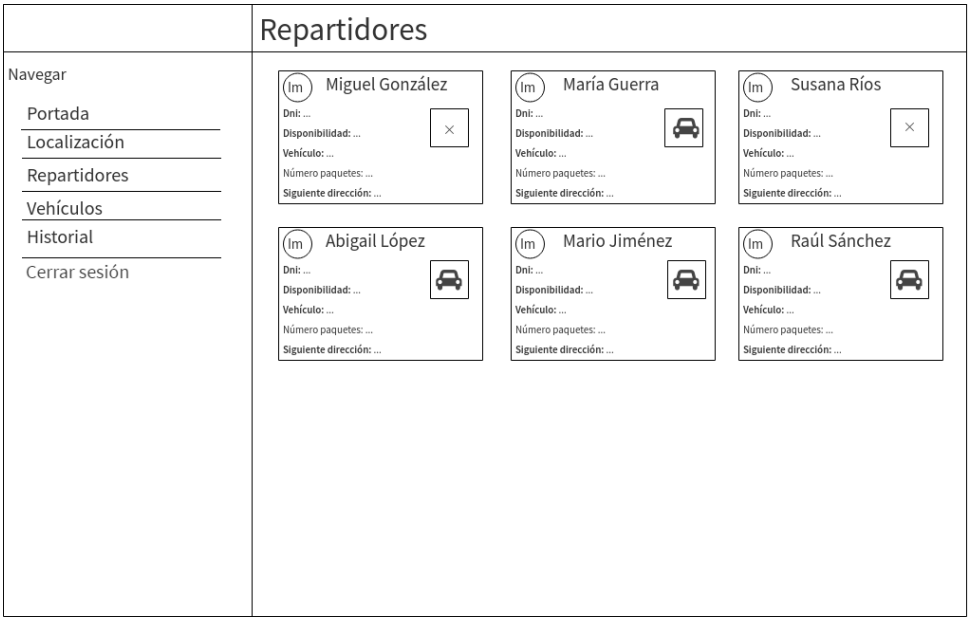

Imagen VII: Página de repartidores

Seleccionando cualquiera de las tarjetas anteriores, se puede acceder a la información detallada de cada repartidor, tal y como se muestra en la *Imagen VIII*.

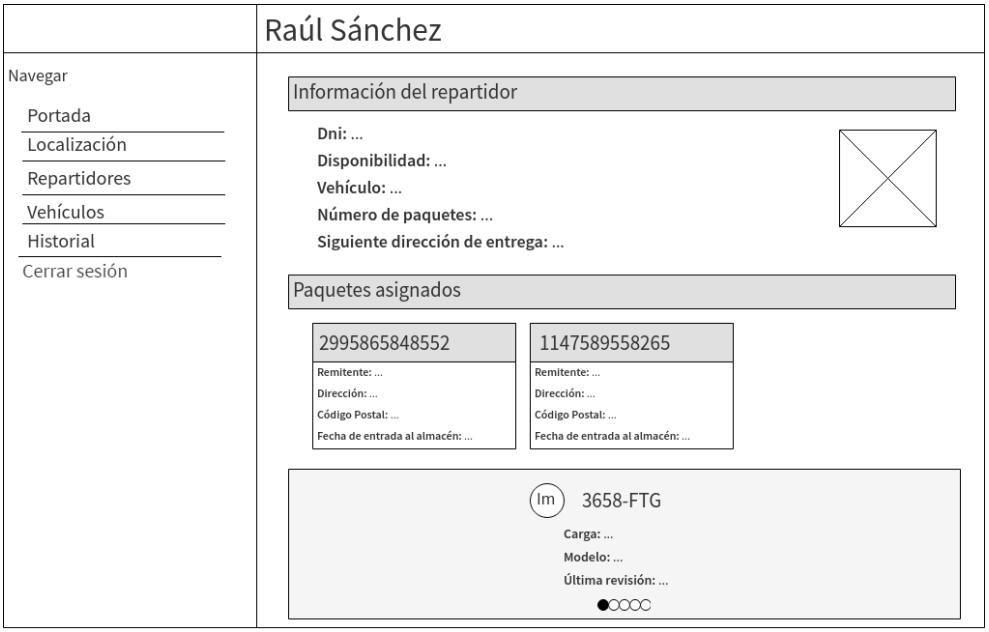

Imagen VIII: Página individual del repartidor

Como se observa, este repartidor no posee un vehículo asignado, por lo que aparece un slider con los vehículos que están disponibles, permitiendo asignarle cualquiera de ellos. En el caso de que ya el repartidor tuviera asignado uno, el slider se sustituiría por un botón para desasignar el vehículo.

En la *Imagen IX* se muestra la página donde se gestionan los vehículos. Se puede acceder a ella desde la opción “Vehículos” del menú lateral.

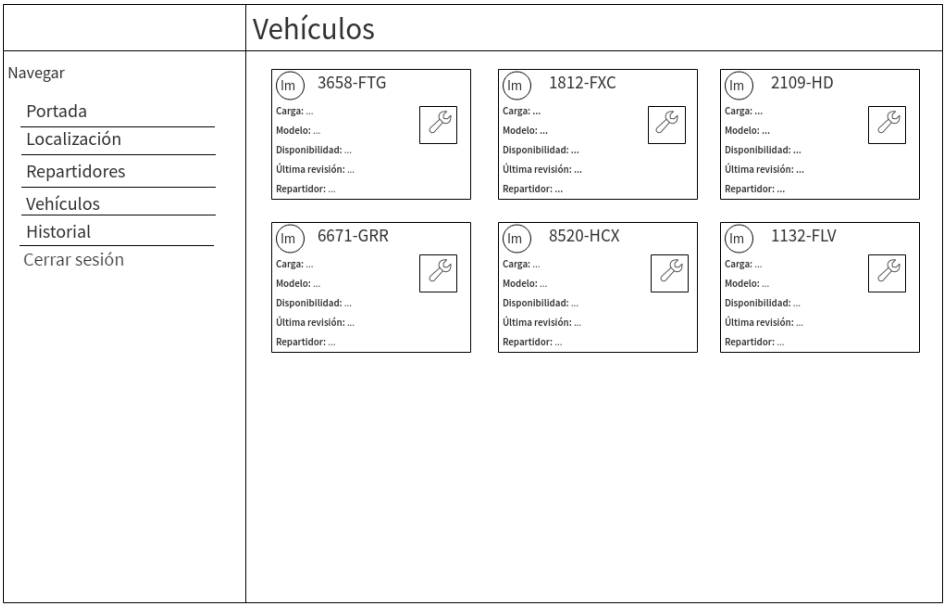

Imagen IX: Página de vehículos

Por último, la *Imagen X* muestra el historial de paquetes entregados (opción “Historial”), el cual mostrará una lista de todos los paquetes que han sido entregados ordenador por fecha de entrega.

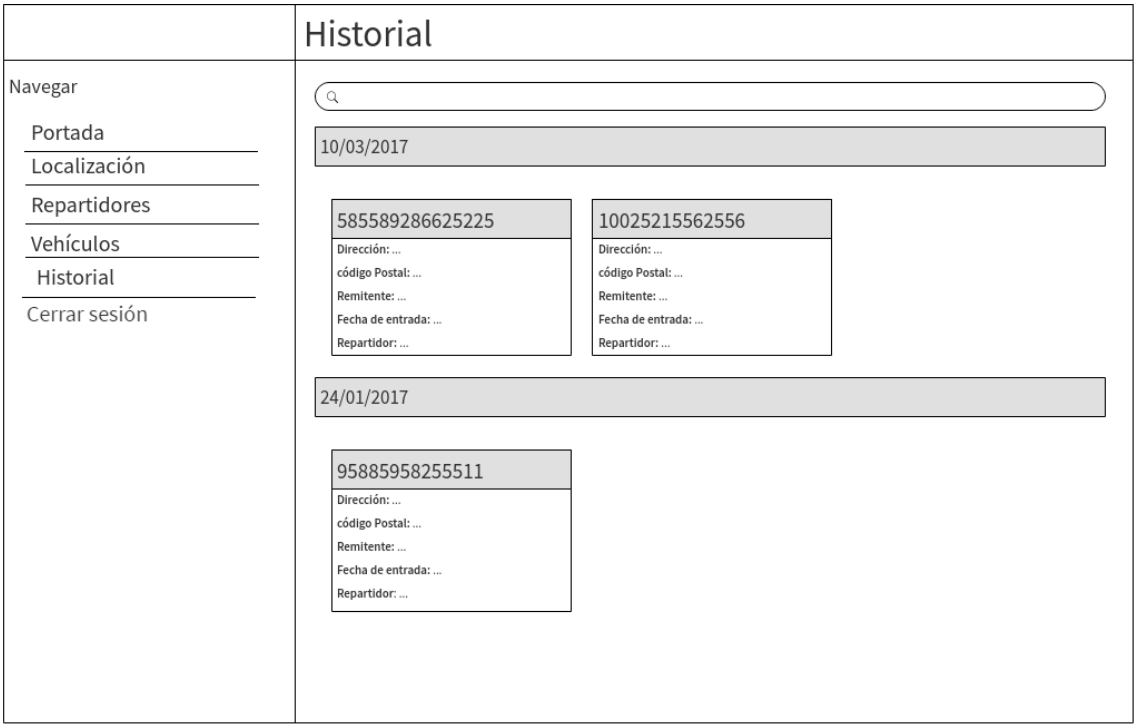

Imagen X: Página Historial

A medida que se iban implementando todos los detalles de la aplicación especificados en estos bocetos, surgía la necesidad de hacer cambios en la estructura de la interfaz de ciertas páginas, tanto por la aparición de problemas como por la de nuevas ideas. Por ello, el resultado final difiere un poco con respecto a la idea inicial de la aplicación.

En este proyecto se ha empleado una paleta de colores basado en el blanco, empleado en los fondos de las páginas, y el rojo, empleado en las cabeceras de las páginas y diversos elementos puntuales. El propósito era obtener una interfaz visualmente atractiva y sencilla ya que debe usarse durante una jornada completa de trabajo, por lo que un tono cálido como el rojo en conjunto con el blanco es perfecto para evitar el cansancio de la vista de los trabajadores que empleen las aplicaciones. El color rojo se ha empleado además para destacar secciones o resaltar algún elemento como separadores de elementos o botones.

Para los textos se ha usado el color negro sobre fondo blanco o texto blanco sobre fondo rojo (como sucede en las cabeceras de secciones), con un tamaño de letra escalable a la longitud de la pantalla. Se ha empleado la negrita para resaltar los títulos de campos informativos de forma que los usuarios que empleen la aplicación encontrarán con facilidad los elementos que estén buscando.

4.5. Modelo de la base de datos

La base de datos empleada en este proyecto es la que ofrece Firebase, de Google. Cuenta con tres tablas diferentes, las cuales contienen toda la información necesaria de los diferentes repartidores, pedidos y vehículos para el correcto funcionamiento de las funcionalidades especificadas del proyecto:

Repartidores

- **UID**: Este campo es usado para almacenar.
- **coche**: Recoge la matrícula del vehículo asignado a ese repartidor.
- **disponibilidad**: Indica si un repartidor está en ruta o se encuentra en el almacén.
- **dni**: DNI del repartidor.
- **horaCapturaGPS**: Este campo almacena el momento en el que se obtuvo la última posición del repartidor.
- **imagen**: Es utilizado para almacenar la URL de la imagen, almacenada en Firebase.
- **latitud**: Recoge la coordenada de latitud de la posición de un repartidor. Útil para la geolocalización.
- **longitud**: Recoge la coordenada de longitud de la posición de un repartidor. Útil para la geolocalización.
- **nombre**: Nombre completo del repartidor.
- **numPedidos**: Número de paquetes asignados a ese repartidor.
- **sigDireccion**: Es la siguiente dirección de entrega a la que irá un repartidor.

Pedidos

- **codigoPostal**: Código Postal del destino del paquete.
- **destinatario**: Nombre del destinatario.
- **dimensiones**: Dimensiones del paquete en centímetros.
- **dirección**: Domicilio donde se debe entregar el paquete.
- **dni**: DNI del usuario que recibirá el paquete.
- **estado**: Indica si un paquete está asignado, en el almacén, entregado, en ruta, etc.
- **fechaEntradaAlmacen**: La fecha en la que ha llegado el paquete al almacén.
- **fechaEntrega**: Fecha en la que se ha entregado un paquete.
- **firma**: Firma del cliente una vez haya recibido el paquete en su domicilio.

- **idPaquete**: Número de identificación de paquete.
- **idRepartidor**: Número de identificación del repartidor al que ha sido asignado el paquete.
- **latitud**: Coordenada de latitud de la dirección donde se debe entregar.
- **longitud**: Coordenada de longitud de la dirección donde se debe entregar.
- **observaciones**: Sirve para indicar si un paquete tiene alguna consideración especial o algún error.
- **peso**: Peso del paquete.
- **remitente**: Entidad que desea enviar el paquete.
- **repartidor**: Nombre del repartidor al que ha sido asociado el paquete.
- **telf**: Teléfono de contacto del cliente que recibe el paquete.
- **urgente**: Sirve para indicar si un cliente ha solicitado envío urgente.

Coches

- **carga**: Capacidad de carga de un vehículo.
- **disponibilidad**: Sirve para indicar si un vehículo está asignado a un repartidor o está libre.
- **imagen**: Contiene la URL de la imagen almacenada en Firebase del vehículo.
- **matricula**: Matrícula del vehículo.
- **modelo**: Modelo del vehículo: Renault, Toyota, etc.
- **repartidor**: Nombre del repartidor al que le ha sido asignado ese vehículo.
- **ultimaRevision**: Fecha de la última revisión que ha pasado ese vehículo.

Como se puede ver, las tablas Vehículos y Repartidores están relacionadas por los campos "*matricula*" (en Vehículos) y "coche" (en Repartidores). Ambos campos contienen la matrícula de vehículo asignado.

Por otro lado, las tablas Repartidores y Pedidos están relacionadas a través de los campos "idRepartidor" (en Pedidos) y la clave única de cada repartidor, generada cada vez que es creado uno nuevo.

4.6. Alcance de la implementación

Debido a la ambición de este proyecto, pese a realizarse en conjunto, el número de horas necesarias para completarlo supera con creces las 300 horas máximas que establece el reglamento. Por ello, se ha optado por construir un prototipo de la aplicación que tendrá la mayor parte de las funcionalidades descritas anteriormente, pero con algunas limitaciones.

Antes de comenzar describiendo las funcionalidades que se han omitido en el proyecto, hay que destacar que la implementación del servidor de almacenamiento y base de datos se ha realizado en la nube, aprovechando el servicio Firebase de Google. Esto permite disponer de la información en tiempo real en cualquier localización, a una velocidad bastante aceptable y con una tasa de transferencia adecuada para un proyecto de estas características. Es de suponer que una empresa que quisiera emplear las aplicaciones que forman este proyecto pondría en funcionamiento un servidor dedicado.

La principal funcionalidad que no se ha implementado es la de la comunicación entre administrador y repartidores, debido a las exigencias en la seguridad de las comunicaciones. A fin de proteger las conversaciones mantenidas, los mensajes deberán cifrarse de extremo a extremo, sin posibilidad de ser descifrados en su paso por el servidor. Este cifrado requiere una implementación tan elaborada en el software del emisor y el receptor que hace inviable su desarrollo en un proyecto con las limitaciones de infraestructura, tiempo y recursos humanos actuales.

Entre las funcionalidades implementadas con limitaciones se encuentra la asignación de pedidos a repartidores, más concretamente en la posibilidad de asignar múltiples paquetes a un repartidor. Debido al estilo empleado en la página principal, introducir esta tarea implicaría rediseñar dicha página principal y reordenar los elementos que la forman, lo cual repercutiría negativamente en el tiempo de desarrollo. A esta situación es preciso añadir la posibilidad de ordenar los elementos del historial de pedidos, cuya implementación tampoco se ha llevado a cabo.

El resto de funcionalidades se han implementado completamente, si bien hay ligeras variaciones respecto al objetivo deseado, los requisitos se han cumplido satisfactoriamente.

4.7. Requisitos hardware mínimos

La versatilidad que ofrece Ionic garantiza su funcionamiento en cualquier dispositivo de tipo PC, pero es recomendable emplear un equipo más potente para sacar partido a esta tecnología. El equipo utilizado cuenta con un procesador Pentium a 2.10 GHz; 4 GB de memoria RAM; Windows 7 Professional.

4.8. Herramientas de desarrollo

Las herramientas y librerías que se emplearán son:

Icono	Nombre	Descripción	Versión
	Ionic	Esta tecnología es un *framework* de código abierto empleado en el desarrollo de aplicaciones híbridas móviles. Creado sobre "AngularJS" y "Apache Cordova", "Ionic" proporciona las herramientas y servicios necesarios para, mediante el uso de HTML y CSS3, desarrollar aplicaciones para móviles sin emplear el lenguaje nativo de cada plataforma.	3.1.1
	Angular	Es un *framework* de código abierto desarrollado por Google y es utilizado para crear y mantener aplicaciones web. Su principal objetivo es aumentar las aplicaciones basadas en navegador con capacidad de "Modelo Vista Controlador". Se emplea en combinación con un entorno de ejecución como NodeJS.	4.0.2
	TypeScript	Se trata de un lenguaje de programación de código abierto desarrollado por Microsoft. Es un conjunto de JavaScript, pero añadiendo tipado estático y objetos basados en clases.	2.2.1
	HTML5 y CSS3	En primer lugar, tenemos HTML5, un lenguaje de marcado que se encuentra en su quinta revisión. Este lenguaje es empleado en la "World Wide Web". A continuación, tenemos CSS3, una hoja de estilo en cascada que se encuentra en su tercera revisión, siendo empleado para establecer el diseño visual de las páginas web e interfaces de usuario.	HTML5 + CSS3
	Cordova	Apache Cordova es un marco de desarrollo móvil de código abierto. Permite utilizar las tecnologías estándar web como HTML5, CSS3 y JavaScript para desarrollo multiplataforma, evitando el lenguaje de desarrollo nativo de cada plataforma móvil.	6.5.0
	NodeJS	Se trata de un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. NodeJS usa un modelo de operaciones E/S sin bloqueo y orientado a eventos, que lo hace liviano y eficiente. El ecosistema de paquetes de Node.js, "npm", supone el conjunto más grande de librerías de código abierto en el mundo.	7.7.0
	Visual Studio Code	Editor de código empleado para la realización del proyecto, destaca por su sencillez de uso, personalización y capacidad de integrar extensiones que aumenten sus funcionalidades.	1.13.1
	Lodash	Empleado para facilitar el uso de JavaScript, emplea métodos para iterar listas, objetos y cadenas de texto, además de manipular valores y crear funciones complejas.	4.17.4

Typings The TypeScript Definition Manager	Typings	Puesto que JavaScript por sí mismo no emplea tipos de datos, es necesario crear ficheros que describan los tipos empleados en nuestro código que pertenezcan a librerías externas. Typings se encarga de instalar los ficheros descriptivos necesarios de dichas librerías.	2.1.1
	AngularFire2	Proporciona una forma de comunicación entre Angular 2 y Firebase. La creación de objetos observables, la sincronización de objetos en tiempo real o la autenticación de usuarios son sus grandes virtudes.	2.0.0-beta.8
AGM ANGULAR GOOGLE MAPS	Angular Google Maps	Esta librería ofrece una total integración de los componentes de Google Maps en Angular 2.	1.0.0-beta.0
Moment.js	Moments	La librería Moment.json permite realizar operaciones con fechas y horas en JavaScript.	2.18.1

4.9. Diseño e implementación

4.9.1. Arquitectura del software

La creación del proyecto genera un árbol de carpetas y ficheros que permite organizar el código. A continuación, se puede observar la utilidad de cada uno de estos elementos:

- **node_modules/** - En este directorio se encuentran todos los módulos necesarios para el correcto funcionamiento de Ionic. La instalación del proyecto incluye estos módulos.
- **Platforms/** - Este directorio contiene las plataformas para las que se ha construido el proyecto. En sus respectivas carpetas se almacenará una versión del proyecto actual para cada plataforma. En este caso, se encuentra la versión para Android y para navegadores.
- **plugins/** - Los plugins que se instalen en este proyecto se encuentran en esta carpeta.
- **resources/** - Tanto el icono e imagen de carga de la aplicación, como los iconos relativos a las versiones de Android e iOS se encuentran en este directorio.
- **src/** - En esta carpeta se encuentran las páginas de la aplicación, así como el estilo que tendrán o el control de los módulos que la integran.
    - **app/** - Este directorio contiene los ficheros principales del proyecto, tales como el fichero de que maneja la página principal de la aplicación (y el menú lateral) o el fichero que gestiona los módulos que se incluirán a la hora de ejecutar la aplicación.
    - **assets/** - En este directorio es empleado para almacenar algunos recursos por deseo del desarrollador, como algunas imágenes que quiera introducir en la aplicación. En este caso, se ha utilizado para incluir iconos y los marcadores para los mapas.
    - **pages/** - Aquí se encuentran las carpetas que contienen las páginas que forman la aplicación. Cada carpeta contiene un fichero de tipo ".html", otro de tipo ".ts" y en ocasiones uno de tipo ".scss"
        - **deliverer/** - Contiene la página que actúa como una de las pestañas del repartidor. En este caso, se muestra información sobre el repartidor

seleccionado y los pedidos seleccionados para entregar, además de un mapa de las localizaciones de entrega de dichos pedidos.
    - **delivererHome/** - Contiene la página que forma un marco con tres pestañas para la información del repartidor seleccionado, sus pedidos asignados y los pedidos entregados.
    - **delivererLocation/** - Contiene la página que muestra la localización de todos los repartidores en tiempo real.
    - **delivererPackFinish/** - Contiene la página que actúa como una de las pestañas del repartidor. En este caso, muestra los paquetes entregados por el repartidor seleccionado.
    - **delivererPacks/** - Contiene la página que actúa como una de las pestañas del repartidor. En este caso, muestra los paquetes asignados al repartidor seleccionado.
    - **delivery-men/** - Esta carpeta contiene la página que muestra un listado de repartidores, con enlaces hacia la página de cada uno.
    - **home/** - Contiene la página principal de la aplicación, dividida en tres segmentos: Paquetes sin asignar (en el almacén); Paquetes asignados a repartidores; Paquetes con incidencias. Se muestra un mapa de la localización de los envíos y un buscador de paquetes.
    - **login/** - Esta carpeta contiene la página de inicio de sesión en la aplicación.
    - **map/** - Esta carpeta contiene la página que muestra el mapa con la dirección de entrega del paquete seleccionado.
    - **order/** - En esta carpeta se encuentra la página que muestra información del pedido seleccionado, así como el control para asignarlo a un repartidor.
    - **registry/** - Este directorio contiene la página que actúa como historial de los paquetes entregados. El historial está ordenado por fecha, además de disponer de un buscador de paquetes.
    - **vehicles/** - En esta carpeta se encuentra la página que muestra un listado de los vehículos de la empresa, así como datos sobre dichos vehículos y la asignación o no a un repartidor.
    - **pages.ts –** Este fichero contiene un índice de las clases creadas en cada página, lo que permite que, en caso de ser necesario, ser importado en una página que quiera trabajar con alguna otra página, tal como navegar hacia esa página.
  - **providers/** - Esta carpeta alberga el fichero “auth-data.ts”, que contiene la configuración de control de la acción de inicio y cierre de sesión.
  - **theme/** - En esta carpeta se encuentra el fichero “variables.scss”, que contiene valores introducidos por el desarrollador sobre el estilo de la aplicación.
  - **validators/** - Este directorio contiene el fichero “email.ts”, que se encarga de la validación del correo introducido a la hora de iniciar sesión en la aplicación.
- **typings/** - Esta carpeta contiene los ficheros importados por la librería “Typings” de los módulos que requieren un tipo no propio de TypeScript.
- **www/** - Contiene el código necesario para la simulación de la aplicación en el navegador, sin necesidad de construir el proyecto.

- **.gitignore** - Este fichero contiene las carpetas y archivos que no estarán controlados por Git.
- **config.xml** - Este fichero alberga información sobre la aplicación, tal como el nombre de la misma, descripción, iconos de la aplicación (para su visualización en un "Smartphone"), preferencias de uso y plugins a importar.
- **package.json** - El fichero contiene el núcleo de Ionic, albergando las dependencias necesarias para ejecutar el proyecto, las librerías instaladas y los plugins añadidos.

4.9.2. Desarrollo del prototipo

Después de analizar de forma exhaustiva el código de la aplicación, se han encontrado ciertas funcionalidades que resaltan con respecto a las demás debido a su importancia. Es el caso de la implementación del mapa, por ejemplo. De las numerosas formas existentes de implementar los mapas en la aplicación, se ha decidido usar la manera más sencilla, rápida y de menor código posible. De manera que, sólo es preciso inicializar el mapa en fichero TypeScript, con los datos correspondientes, como se observa en la *Imagen XI.*

map.page.ts

```
16  ionViewDidLoad(){
17    let order = this.navParams.data;
18
19    this.map = {
20      lat: order.latitud,
21      lng: order.longitud,
22      zoom: 15,
23      markerLabel: order.direccion
24    };
25
26  }
```

Imagen XI: Código de inicialización del mapa

Y, por último, mostrarlo con unas pocas etiquetas HTML y unas líneas de SCSS, tal y como se muestra en la *Imagen XII*.

map.page.html

```
7   <ion-content class="map-page">
8       <agm-map id="map" [latitude]="map.lat" [longitude]="map.lng" [zoom]="map.zoom" >
9           <agm-marker [latitude]="map.lat" [longitude]="map.lng">
10              <agm-info-window>
11                  <strong>{{map.markerLabel}}</strong>
12              </agm-info-window>
13          </agm-marker>
14      </agm-map>
15  </ion-content>
```

map.page.scss

```
1   .map-page #map {
2     width: 100%;
3     height: 100%;
4     transition: opacity 150ms ease-in
5   }
```

Imagen XII: Código HTML y SCSS del mapa

Otro aspecto que llama la atención es la utilización de la función "ionViewDidLoad()", en la cual se extraen los datos de Firebase para luego ser mostrados. El propósito esencial de este método es ejecutarse varias veces, a diferencia del constructor que sólo se ejecuta una vez. Como los datos de Firebase varían, es necesario que se pueda ver esos cambios en la página. Usando este método, no es preciso refrescar la página en busca de actualizaciones, ya que se realizan automáticamente. Por ello, es empleado con frecuencia en las páginas de la aplicación. En la *Imagen XIII* se muestra un ejemplo de la función usada en una de las páginas.

deliverer.page.ts

```
ionViewDidLoad(){
    this.deliverer = this.navParams.data;
    let loader = this.loadingController.create({…
    });

    loader.present().then(() => {
        this.angularFire.database.list('/pedidos').subscribe(data => {
            this.ordersData = _.chain(data)
                               .filter(o => o.idRepartidor === this.deliverer.$key && o.estado == "En reparto" || o.estado == "Siguiente en entrega")
                               .value();
            this.orders = this.ordersData;
            for (var i = 0; i< this.ordersData.length; i++){…
            }
        });

        this.map = {…
        this.angularFire.database.list('/coches').subscribe(data => {
            this.vehiclesData = _.chain(data)
                                 .filter(v => v.disponibilidad === "Libre")
                                 .value();

            this.vehicles = this.vehiclesData;
            this.vehiclesDatabase = this.angularFire.database.list('/coches');
            this.deliveryMen = this.angularFire.database.list('/repartidores');

            loader.dismiss();
        });
    });
}
```

Imagen XIII: Función "ionViewDidLoad()" en la página individual del repartidor, pestaña "Info"

Por otra parte, se hace uso un buscador en varias páginas, que facilite la obtención de información sin tener que perder mucho tiempo buscando por toda la página. Al principio supuso un problema añadir varios campos para el filtrado, ya que las soluciones que se ofrecían eran demasiado complejas. Hasta que se obtuvo un método mucho más sencillo y limpio de realizar filtrados por varios campos, gracias las diferentes operaciones que se permiten realizar en la función ".filter()" de la librería *Lodash*. En la *Imagen XIV* se observan los diferentes campos por los que se filtra en el buscador de una página.

registry.page.ts

```
search(){
    let queryTextLower = this.queryText.toLowerCase();
    let filteredOrders = [];

    _.forEach(this.allDates, dat => {
        let orders = _.filter(dat.order, or => (<any>or).repartidor.toLowerCase()
        .includes(queryTextLower) || (<any>or).fechaEntrega.toLowerCase()
        .includes(queryTextLower) || (<any>or).remitente.toLowerCase()
        .includes(queryTextLower) || (<any>or).idPaquete.toString().includes(queryTextLower));
        if (orders.length) {
            filteredOrders.push({ date: dat.date, order: orders});
        }
    });

    this.orders = filteredOrders;
}
```

Imagen XIV: Función que permite filtrar una búsqueda en la página del historial

Por último, cabe destacar el control de la autenticación por parte de AngularFire y Firebase. Ambos se encargan de comprobar si el usuario introducido está registrado en la parte de Autenticación de Firebase y si la contraseña coincide con el correo electrónico. De esta manera, el desarrollador no tiene que hacer demasiadas comprobaciones, simplemente envía los datos recogidos en el formulario para iniciar sesión y los envía a las funciones de AngularFire, que se encargará de lo demás. En la imagen XV se observa el código de control para la autenticación.

auth-data.ts

```
11    constructor(public angularFire: AngularFire) {
12      angularFire.auth.subscribe( user => {
13        if (user) { this.fireAuth = user.auth; }
14      });
15    }
16
17    getCurrentUid(){
18      return this.angularFire.auth.getAuth().auth.uid;
19    }
20
21    loginUser(newEmail: string, newPassword: string): firebase.Promise<any> {
22      return this.angularFire.auth.login({
23        email: newEmail,
24        password: newPassword
25      });
26    }
27
28    logoutUser(): firebase.Promise<any> {
29      return this.angularFire.auth.logout();
30    }
31
32  }
```

Imagen XV: Código TypeScript de control de autenticación

4.9.3. Pruebas

Se ha llevado a cabo una batería de pruebas para comprobar el correcto funcionamiento de la aplicación. Aunque se ha ido comprobando cada una de las funcionalidades a medida que se implementaban, una vez acabado por completo el prototipo, se han repetido todas las pruebas para corroborar que todo funciona como es debido. En la *Tabla 1* se muestran los resultados de las dos fases de prueba.

Funcionalidades	Primera fase	Segunda fase
Iniciar sesión en la aplicación	Correcto	Correcto
Mapa de todos los paquetes	Conseguir un marcador de distinto color para cada estado del paquete	Correcto
Buscador en Asignados/ Sin Asignar	Filtrar por varios campos	Correcto
Panel lateral	Correcto	Correcto
Asignar repartidor	Correcto	Correcto
Desasignar repartidor	Correcto	Correcto
Añadir incidencia	Sólo para paquetes que no estén asignados a un repartidor	Correcto
Acceder al mapa de cada paquete	Correcto	Correcto
Modificar fecha de última revisión	Correcto	Correcto
Localización de repartidores	Correcto	Correcto
Buscador historial	Filtrar por varios campos	Correcto
Desasignar un vehículo desde la vista general de repartidores	Mejoras en las operaciones con la base de datos	Correcto
Desasignar vehículo desde la vista individual del repartidor	Mejoras en las operaciones con la base de datos	Correcto

Asignar vehículo	Correcto	Correcto
Mapa individual de la posición de un repartidor y la siguiente dirección de entrega	Conseguir un marcador de distinto color para la siguiente dirección	Correcto

Tabla 1: Resultado de las pruebas realizadas

Además, se ha mostrado la aplicación en diferentes pantallas para comprobar su diseño adaptable. En la Tabla 2 se observan los diferentes dispositivos en los que ha sido mostrada:

Dispositivo	Pantalla	Resultado
Samsung Galaxy S5	5.1”	Correcto
Nexus 5X	5.2”	Correcto
Nexus 6P	5.7”	Correcto
Iphone 5	4”	Correcto
Iphone 6	4.7”	Correcto
Iphone 6 plus	5.5”	Correcto
Ipad	9.7”	Correcto
Ipad Pro	10.5”	Correcto
Portátil	15.6”	Correcto
Sobremesa	21.5”	Correcto

Tabla 2: Dispositivos con los que se ha mostrado la aplicación

4.9.4. Acceso al código

Todo el código de la aplicación se puede encontrar en el siguiente repositorio de GitHub: https://github.com/mguerra803/tft-admin-ionic2 .

5. Aspectos económicos y temporales

Las tres aplicaciones del proyecto han sido creadas con vistas a ser empleadas por una posible empresa de transporte de paquetería, de forma que dispongan de un grupo de aplicaciones para agilizar el trabajo del gestor del almacén, aumentar la transparencia con la que trabaja el repartidor y, finalmente, informar debidamente al receptor de un paquete.

Para poner en marcha este proyecto en un entorno realista, es necesario en primer lugar realizar un análisis exhaustivo de los requisitos de cada aplicación, su diseño y posterior codificación. Sin embargo, debido a nuestra falta de experiencia laboral y a que no hemos visto nada similar a lo largo del curso, no podemos estimar el número de horas necesarias para el desarrollo de la aplicación, así como tampoco podemos deducir el coste que conllevaría.

Sabemos que será necesario un servidor para almacenar la información que generen y controlen las aplicaciones. Actualmente, el prototipo está usando la versión gratuita de Firebase. No obstante, esto no sería suficiente en el caso de una aplicación totalmente funcional y en uso comercial. Sería preciso pues, usar una versión de pago de Firebase. Accediendo a su página oficial vemos que Firebase ofrece dos planes de pago: El plan Flame y el plan Blaze. Llevando a cabo una comparativa entre los dos, nos damos cuenta de que el plan Flame, siendo la opción más económica, se podría quedar un poco corto en muchos aspectos en empresas de reparto de gran tamaño, mientras que el plan Blaze, con un precio más elevado, es totalmente ajustable y además ofrece copias de seguridad automáticas.

Como es bien sabido, este negocio ya no consiste en vender nuestra aplicación a una empresa y desvincularnos de ella. Hoy en día el negocio está en ofrecer nuestro servicio a varias empresas y encargarnos del mantenimiento de éste, con todo el gasto que ello suponga. Si no tenemos en cuenta la posible personalización que ofrecerá la aplicación a la empresa, la cual sólo se pagaría una vez, las empresas que quieran hacer uso de la aplicación pagarían una cuota mensual por el mantenimiento. El precio de esa cuota no se ha podido calcular por la misma razón expuesta anteriormente.

Visto todo lo anterior, hay que destacar también el valor añadido que proporciona la transparencia y acercamiento al público, gracias al uso de una aplicación para clientes diseñada para mostrar el máximo de información posible de una forma agradable e intuitiva.

6. Normativa y legislación

Este proyecto se ha desarrollado con especial atención en la fiabilidad, seguridad y calidad de ella, conforme a los principios de desarrollo ético del software y la legislación vigente.

Es preciso tener en consideración una serie de medidas legales debido a que se manejarán datos de carácter personal. El proyecto debe cumplir con el Reglamento de Seguridad del Real Decreto 994/1999 de la Ley Orgánica de Protección de Datos (LOPD). Dicha ley tiene como objetivo garantizar y proteger, en lo concerniente al tratamiento de los datos personales, las libertades públicas y los derechos fundamentales de las personas físicas, atendiendo especialmente a su honor e intimidad personal y familiar.

Dicha ley será aplicada a los datos de carácter personal registrados sobre cualquier tipo de soporte (informático o no). Están excluidos datos recogidos para uso doméstico y material clasificado por el estado que tratan sobre delincuencia organizada o terrorismo y a toda modalidad de uso posterior de estos datos por los sectores público y privado. Si la LOPD no estuviera en vigor, nuestros datos estarían en manos de la especulación.

La LOPD cataloga los ficheros según la naturaleza de los datos que contienen. Acorde con ello, se establecen diferentes niveles de seguridad que garanticen en mayor o menor medida la confidencialidad e integridad de dicha información. Los niveles en los que puede catalogarse un fichero son los siguientes:

- **Nivel alto:** Se considerarán de este nivel aquellos ficheros que contengan datos relativos a la ideología, afiliación social, religión, creencias, origen racial, salud o vida sexual. También forman parte de este nivel los ficheros recabados con fines policiales sin consentimiento de las personas afectadas o aquellos derivados de actos de violencia de género.

- **Nivel medio:** Dentro de este nivel se encuentran aquellos ficheros que contengan relativos a la comisión de infracciones administrativas o penales; aquellos cuyo funcionamiento se rija por el artículo 29 de la Ley Orgánica 15/1999 del 13 de diciembre; prestación de servicios de administraciones tributarias, relaciones con entidades financieras, gestorías o servicios comunes de seguridad social. Así mismo, formarán parte de este nivel aquellos ficheros que sean responsables las mutuas de accidentes de trabajo y enfermedades profesionales de la Seguridad Social; aquellos que contengan un conjunto de datos de carácter personal que ofrezcan una definición de las características o de la personalidad de los ciudadanos y que

permitan evaluar determinados aspectos de la personalidad o del comportamiento de los mismos. Finalmente, este nivel incluye aquellos datos que permitan la localización de una persona.

- **Nivel básico:** Las medidas de seguridad que se han de adoptar en los ficheros que contengan datos de carácter personal están en función de la tipología de tales datos. Al operar de forma "acumulativa", todos los ficheros o tratamientos de datos de carácter personal deberán adoptar las medidas de seguridad calificadas de nivel básico.

Este proyecto hace uso de varios datos de carácter personal de los repartidores de la empresa (como su ubicación exacta) y de los clientes (dirección, teléfono, nombre, DNI, etc.). Por ello, se cuenta con el fichero de datos de los repartidores, que requiere una protección de nivel medio, y el fichero de datos de los clientes, el cual requiere una protección de nivel básico.

A fin de cumplir la normativa vigente, los clientes tendrán a su disposición una serie de condiciones para las cuales se solicita expresamente su conformidad, relativas a la manipulación de sus datos y la debida protección por parte de la empresa. En el Anexo 2 se encuentra el modelo de la Guía de Seguridad de Datos.

7. Conclusiones y trabajos futuros

Los resultados obtenidos tras la finalización del proyecto han sido muy positivos, el prototipo resultante tiene una gran calidad y un desarrollo muy avanzado puesto que se han implementado casi la totalidad de las funcionalidades propuestas. Los objetivos que se perseguían con este proyecto pueden considerarse alcanzados con soltura. Es importante destacar que la estructura del proyecto y la naturaleza de las tecnologías empleadas (que se encuentran en constante actualización) permitirán mejorar ampliamente el trabajo desarrollado, suponiendo además un importante punto de partida en el campo de la programación de aplicaciones móviles.

La realización del proyecto en Ionic 3 y Angular 4 ha supuesto un reto en el desarrollo, debido en primer lugar al desconocimiento inicial de estas tecnologías, pero posteriormente debido a la novedad del "framework" y la falta de documentación disponible, a lo que habría que sumar las constantes actualizaciones de los desarrolladores de Ionic para añadir nuevas funcionalidades. Pese a ello, trabajar con estas tecnologías ha sido ampliamente instructivo e incluso podría decirse que ha sido ameno, gracias principalmente a que el desarrollo del proyecto llegó a convertirse en un entretenimiento y no una carga o pérdida de tiempo. Este hecho nos permitió trabajar con una motivación completamente diferente, planteando nuevas ideas con mayor facilidad, resolviendo problemas en el código con relativa soltura y avanzando con paso firme. En definitiva, la elección de la tecnología sobre la que trabajar, una con la que te encuentres cómodo, es capaz de mejorar la productividad de cualquier trabajador, como ha sucedido en este caso. Por suerte, Ionic cuenta con una comunidad de programadores muy fiel y entregada, lo que augura un futuro muy prometedor para el desarrollo de aplicaciones móviles.

La envergadura del proyecto obligó a separar el planteamiento inicial en dos proyectos distintos, pero, aun así, la interconexión de las aplicaciones obligaba a mantener una comunicación constante para que el delicado trabajo en paralelo requerido tuviera un funcionamiento correcto durante la fase de análisis, creación del diseño inicial y posterior implementación del prototipo. Lo que a simple vista parecía un problema de difícil solución terminó como una colaboración

satisfactoria para ambas partes y con estupendos resultados en el prototipo final. La experiencia obtenida durante las fases de trabajo en paralelo puede considerarse de un valor incalculable y jugará un papel muy importante en el futuro laboral.

Con respecto al futuro del prototipo es perfectamente posible, partiendo de una reconstrucción total en términos de objetivos, funcionalidades y diseño, ampliar el trabajo realizado hasta el punto de obtener una aplicación perfectamente compatible y funcional en el mundo empresarial. Como se detalló en la sección dedicada a los aspectos económicos, la aplicación podría incrementar el volumen de facturación de una empresa que la emplease, partiendo de la base de que el incremento del rendimiento y fluidez en el desempeño de las tareas de los agentes que forman la empresa mejore sustancialmente. Gracias a la versatilidad que proporciona Ionic, no necesariamente tiene que tratarse de una aplicación para una empresa de reparto de paquetes, sino que pueden desarrollarse aplicaciones específicas de control de flotas o de contratación de servicios de transporte de personas (integrar el colectivo de los taxistas en el entorno tecnológico actual), entre muchas posibilidades. En cualquier caso, se puede afirmar que habrá una mejora en el desempeño económico de la empresa que financie el proyecto.

Planteando un futuro más individual, desligado del entorno empresarial, el desarrollo del proyecto ha supuesto una ampliación de conocimientos en desarrollo y diseño tanto web como para dispositivos móviles. Estos conocimientos adquiridos serán especialmente útiles para pequeños proyectos por cuenta propia o incluso en un entorno de trabajo que, si bien no esté interesado en proyectos de esta índole, si puede requerir conocimientos avanzados en estos campos anteriormente mencionados.

Para finalizar, a nivel personal este proyecto ha repercutido en mi forma de trabajar, en la forma de enfrentar un problema y orquestar su solución para seguir adelante. Cada solución, por pequeña que fuera acrecentaba mi motivación por el proyecto, aprender un poco más cada vez de forma que, al terminar el desarrollo, los problemas más complicados no lo eran tanto. No todo fueron problemas, trabajar con Ionic ha resultado sorprendentemente entretenido para tratarse de un proyecto de final de carrera. Desglosadas las funcionalidades de la aplicación en tareas de menor envergadura, cada una presentaba un reto mental bastante atractivo que podría compararse con un rompecabezas simple, casi convirtiendo el desarrollo en un juego. Cada avance te instaba a continuar, a resolver el siguiente puzzle, poniendo a prueba tu ingenio, jugando con el HTML, ajustando el CSS pixel a pixel y preguntándote cómo es posible que TypeScript tenga tantos trucos para encontrar la misma solución. Como dice el dicho, “Escoge un trabajo que te guste y nunca tendrás que trabajar ni un solo día de tu vida”.

8. Fuentes de información

Listado de enlaces consultados durante la realización del proyecto.

- **Preparación de Ionic**
  - Instalación de NodeJS: https://nodejs.org/es/
  - Guía de instalación de Ionic: http://ionicframework.com/docs/guide/installation.html
  - Documentación de Ionic: https://ionicframework.com/docs/
  - GitHub de Ionic: https://github.com/ionic-team/ionic/blob/master/CHANGELOG.md
  - Documentación de Visual Studio Code: https://code.visualstudio.com/docs/?dv=win

- **Librerías y herramientas empleadas**
  - GitHub de AngularFire2: https://github.com/angular/angularfire2
  - GitHub de Angular Google Maps: https://github.com/SebastianM/angular-google-maps
  - Documentación de Moment.js: http://momentjs.com/docs/
  - Utilidad para generar códigos QR: http://www.qrcode.es/es/generador-qr-code/
  - Marcadores para Google Maps: https://github.com/Concept211/Google-Maps-Markers

- **Otros**
  - Curso de Ionic 2, Angular 2 y TypeScript: https://www.pluralsight.com/courses/ionic2-angular2-typescript-mobile-apps
  - Precios Firebase: https://firebase.google.com/pricing/?hl=es-419

9. Anexos

ANEXO I: Manual de usuario y software

Esta aplicación está pensada para la gestión de paquetes, repartidores y vehículos de una empresa de reparto de forma rápida e intuitiva, facilitando el trabajo del gestor del almacén. A continuación, se expone el manual de usuario de esta aplicación, detallando cada una de las funcionalidades que ofrece.

En la *Ilustración 1* vemos la primera página que aparece al iniciar la aplicación. Debemos ingresar el correo electrónico y la clave para acceder a ella.

Ilustración 1: Página de acceso a la aplicación

Una vez iniciada la sesión, se muestra la página principal (*Ilustración 2*), encargada de gestionar los paquetes que se registran en el almacén.

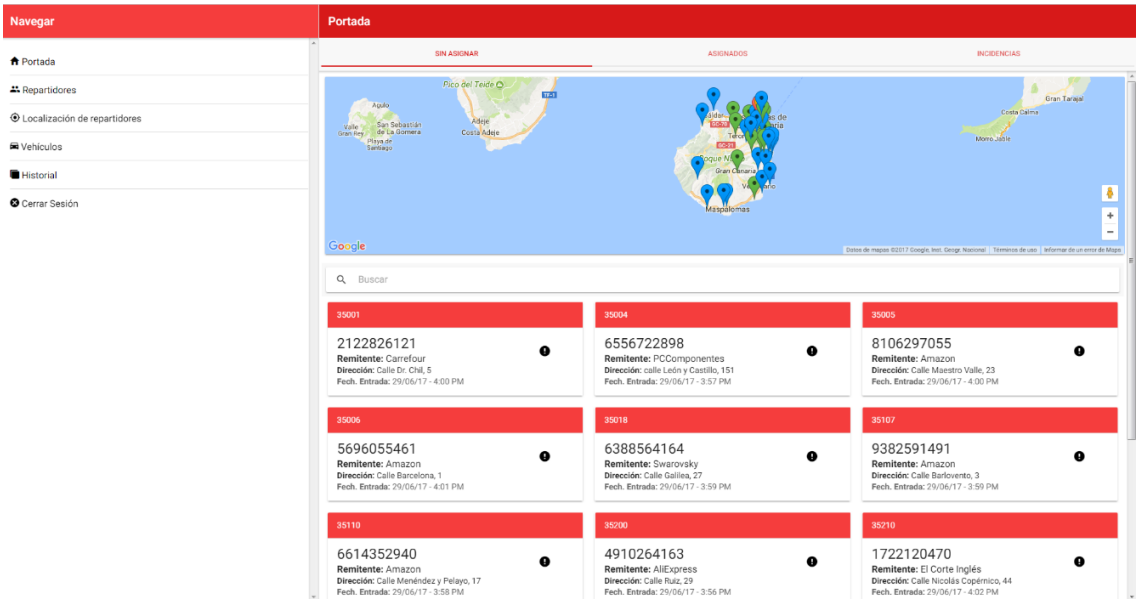

Ilustración 2: Página principal, segmento "SIN ASIGNAR"

Esta página se divide en tres segmentos: “SIN ASIGNAR”, “ASIGNADOS” e “INCIDENCIAS”. El primer segmento, que se muestra en la *Ilustración 2*, es el que se muestra por defecto y en él podemos ver los paquetes que aún no han sido asignados a un repartidor, agrupados por códigos postales. El segundo segmento muestra los paquetes que ya han sido asignados y se agrupan por repartidores, tal y como se observa en la *Ilustración 3*.

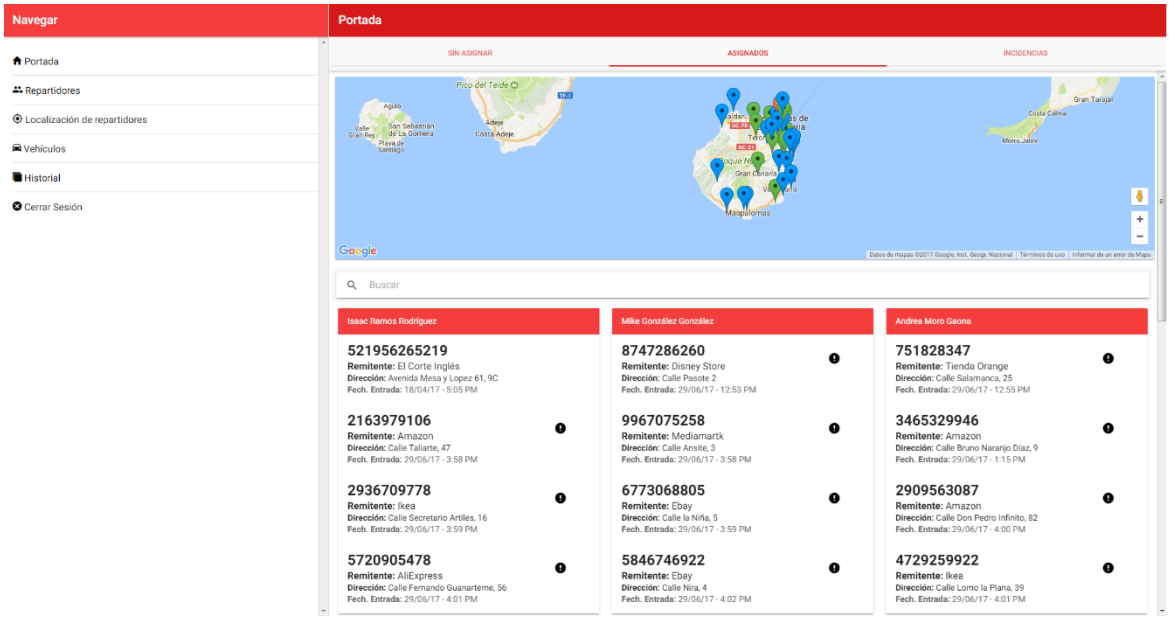

Ilustración 3: Página principal, segmento “ASIGNADOS”

Seleccionando el último segmento podemos ver una lista de los paquetes que tienen alguna incidencia (*Ilustración 4*).

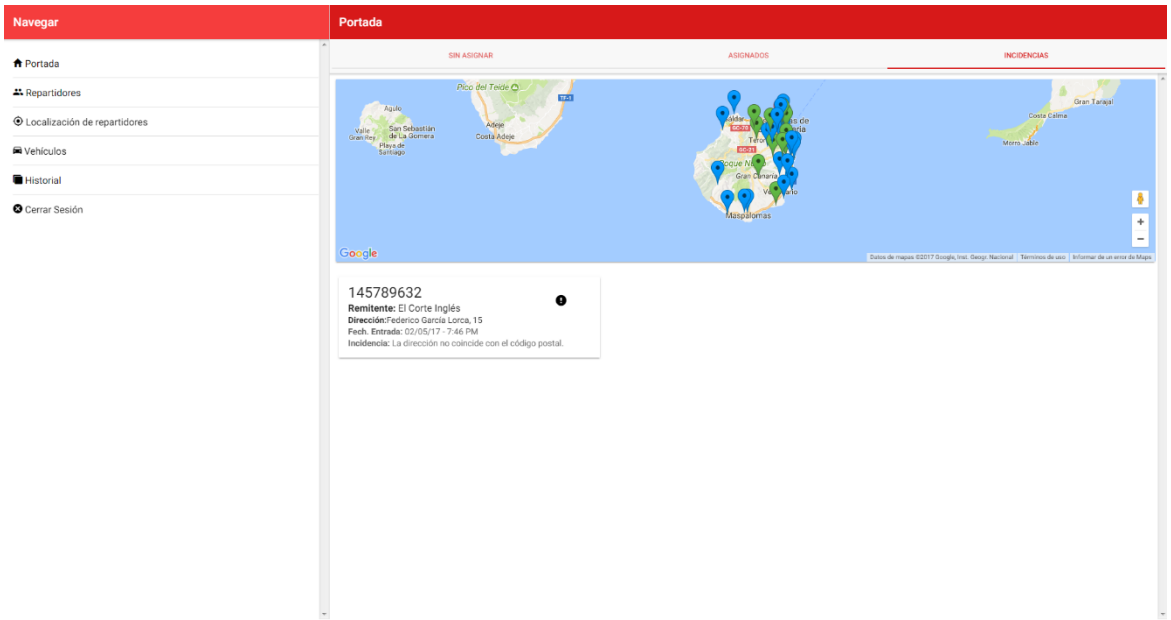

Ilustración 4: Página principal, segmento “INCIDENCIAS”

En todos los segmentos se puede ver un mapa en el que se muestran todos los paquetes menos los que ya están entregados y los que tienen alguna incidencia. Cada estado de los paquetes se corresponde con un color de marcador:

- Verde: Paquetes sin repartidor asignado.
- Azul: Paquetes con un repartidor asignado.
- Rojo: Paquetes seleccionados para el reparto.
- Naranja: Paquetes que serán los siguientes en ser entregados.

Con el buscador podemos filtrar los paquetes de la lista por varios campos. Por ejemplo, podemos buscar el nombre de una empresa remitente, una fecha específica de entrada de paquetes al almacén, etc. Sólo se mostrarán aquellos paquetes que en su información tengan algún dato que coincida con lo que estamos buscando.

Podemos acceder a la información de cada paquete de dos formas. La primera es seleccionando alguno de los paquetes de las listas de los segmentos (menos en la lista de paquetes con incidencias) y la segunda forma es seleccionando alguno de los marcadores del mapa y haciendo click en el ID del pedido.

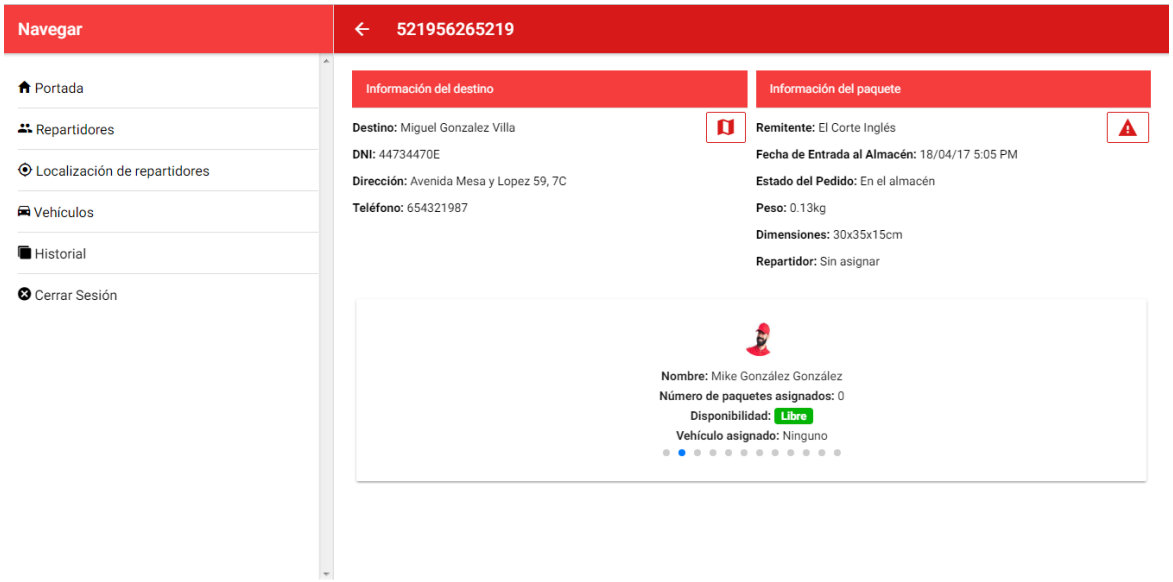

Ilustración 5: Página de información de paquetes

Una vez en la página individual de cada paquete podemos consultar todos sus datos. Además, podemos asignar o desasignar un repartidor a ese paquete. En el caso de la *Ilustración 5*, como el paquete elegido no tiene un repartidor asignado, nos da la opción de elegir uno del slider y asignarlo. Por el contrario, si ya tiene un repartidor asignado, aparecerá un botón que permite desasignarlo.

Po otro lado, vemos que, además de los datos del paquete, tenemos dos botones. El primero de ellos, , nos permite ver la ubicación del domicilio del destinatario en un mapa. El otro botón, , nos permite añadir una incidencia a un paquete, por ejemplo, si vemos que la información de ese paquete es errónea o está incompleta. Esta opción sólo nos aparece si el paquete no tiene asignado ningún repartidor.

Ahora hablaremos de las opciones que aparecen en el menú lateral.

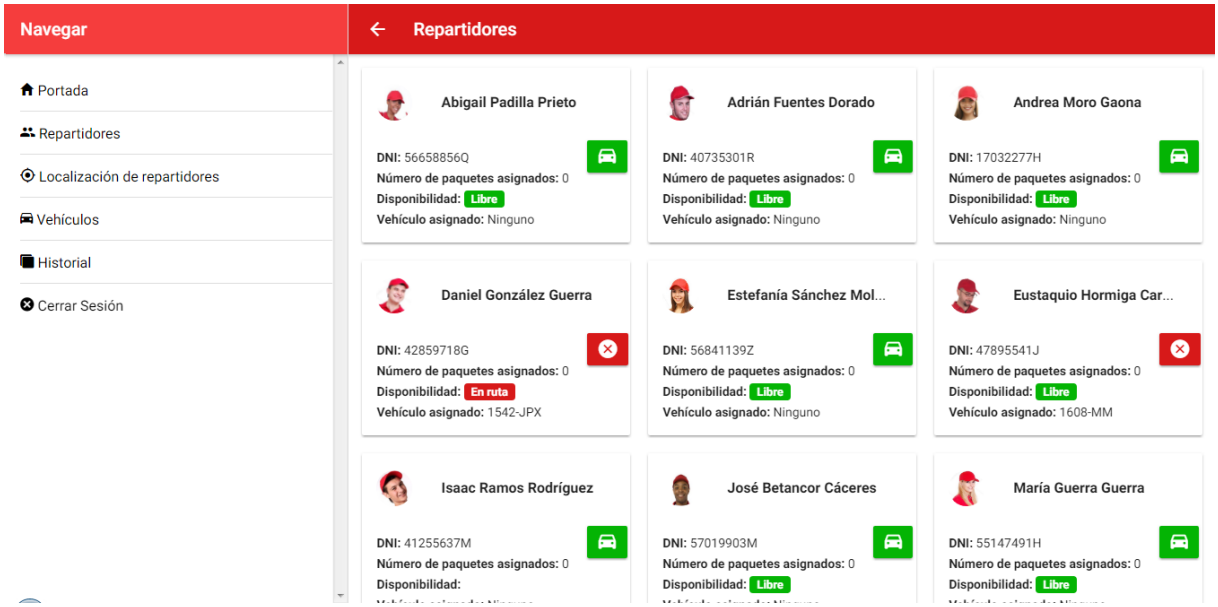

Ilustración 6: Página de repartidores

En "Repartidores" vemos la lista de todos los repartidores de la empresa, ordenada en tarjetas, como se ve en la *Ilustración 6*. Cada una contiene algunos datos del repartidor y un botón, que es diferente según un repartidor tenga asignado un coche, , que también permite desasignar el vehículo directamente, o no tiene vehículo asignado todavía, . Si seleccionamos un repartidor, podemos acceder a su información detallada.

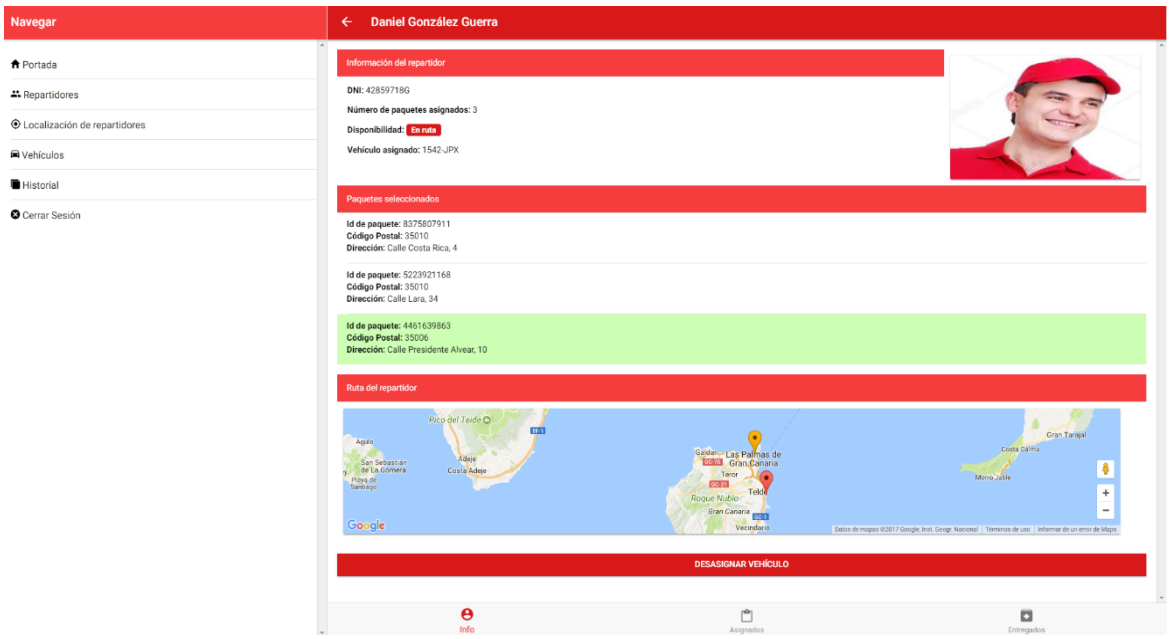

Ilustración 7: Página individual de repartidores, pestaña "Info"

Como vemos en la *Ilustración 7*, esta página está dividida en tres pestañas. En la primera pestaña Info vemos los datos del repartidor y, si no tiene un vehículo asignado, aparecerá una lista con todos los que están disponibles, que pueden ser asignados haciendo click sobre alguno de ellos. Por el contrario, si tiene un vehículo asignado, aparecerá una lista de los paquetes que se han

seleccionado para el reparto de ese día y el mapa de la ubicación del repartidor en tiempo real. El paquete que se muestra en verde en la lista es el siguiente que va a ser entregado. Además, en el mapa se muestra la siguiente dirección de entrega con un marcador naranja.

En la siguiente pestaña 

se listan todos los paquetes asignados a ese repartidor de la misma manera que hemos visto en la portada, pudiendo acceder a ellos de igual forma (*Ilustración 8*).

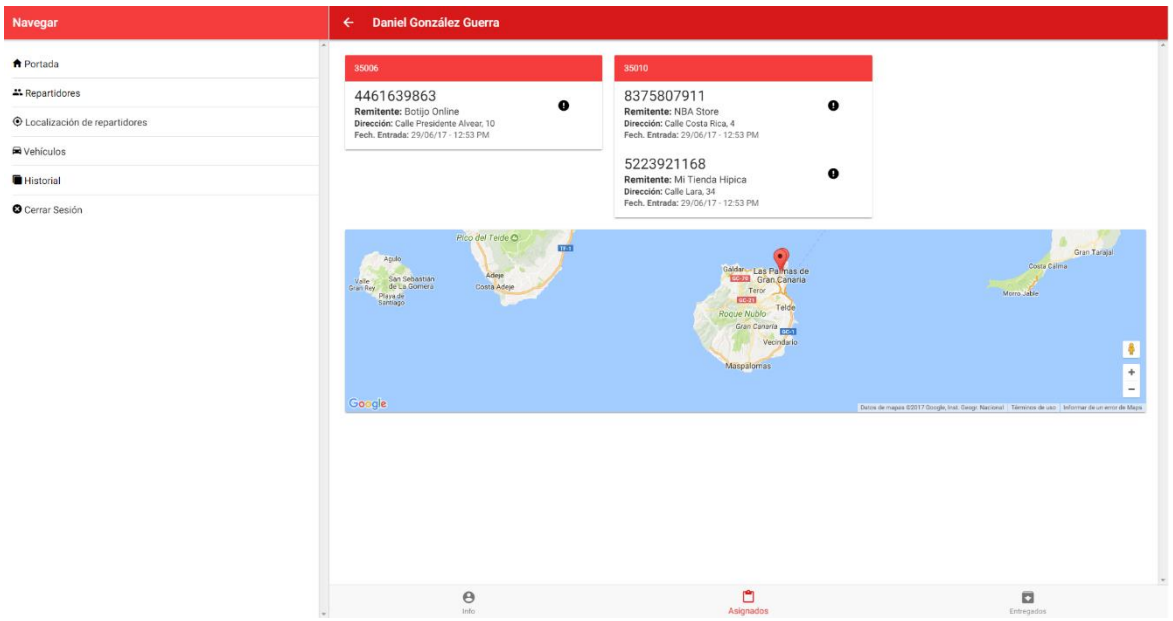

Ilustración 8: Página individual de repartidores, pestaña "Asignados"

En la última pestaña 

vemos los pedidos entregados de ese mismo repartidor, agrupados también por código postal, tal y como se observa en la *Ilustración 9*.

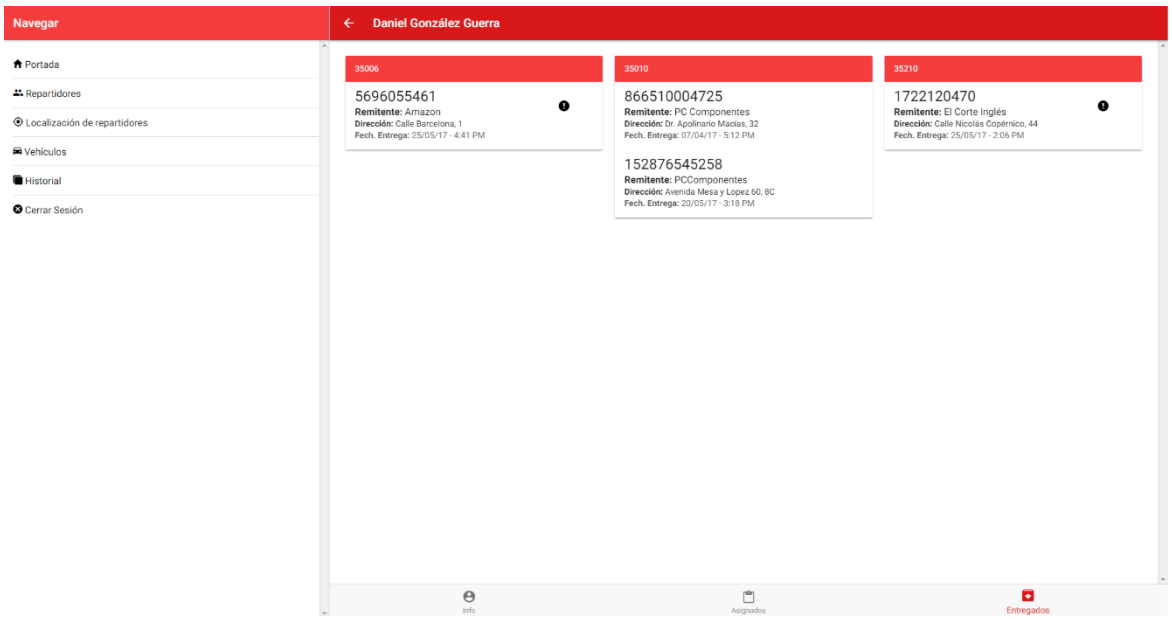

Ilustración 9: Página individual de repartidores, pestaña "Entregodos"

En la página de "Localización de repartidores", como se aprecia en la *Ilustración 10*, podemos ver todos los repartidores que están en ruta en este momento. Aquellos repartidores que aún le quedan paquetes por entregar se verán en color rojo, mientras que los que hayan acabado de repartir todos los paquetes y vuelven al almacén se verán en verde. Podemos acceder a la información de cada repartidor seleccionando el marcador y haciendo click en el nombre.

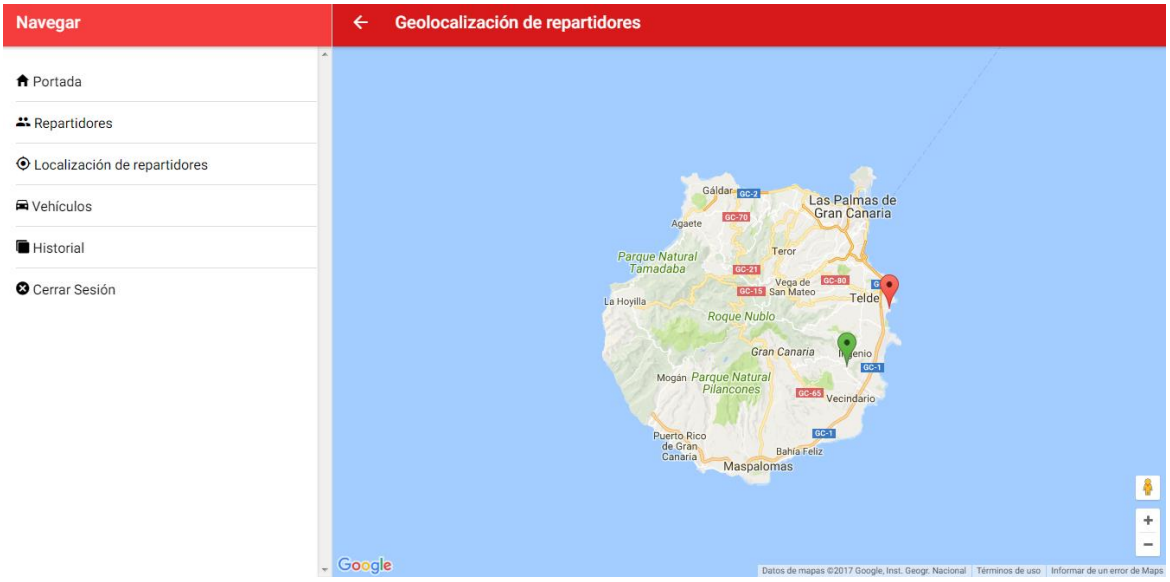

Ilustración 10: Página de localización de repartidores

En "Vehículos" tenemos la lista de los vehículos de los que dispone la empresa organizada en tarjetas (*Ilustración 11*). Cada tarjeta contiene la información relevante de cada vehículo. La disponibilidad nos indica si ese vehículo se encuentra asociado a un repartidor o no. Además, cada tarjeta contiene un botón  para actualizar la fecha de la última revisión del vehículo. Hay que tener en cuenta que la fecha se actualizará al momento en el que ha sido pulsado dicho botón.

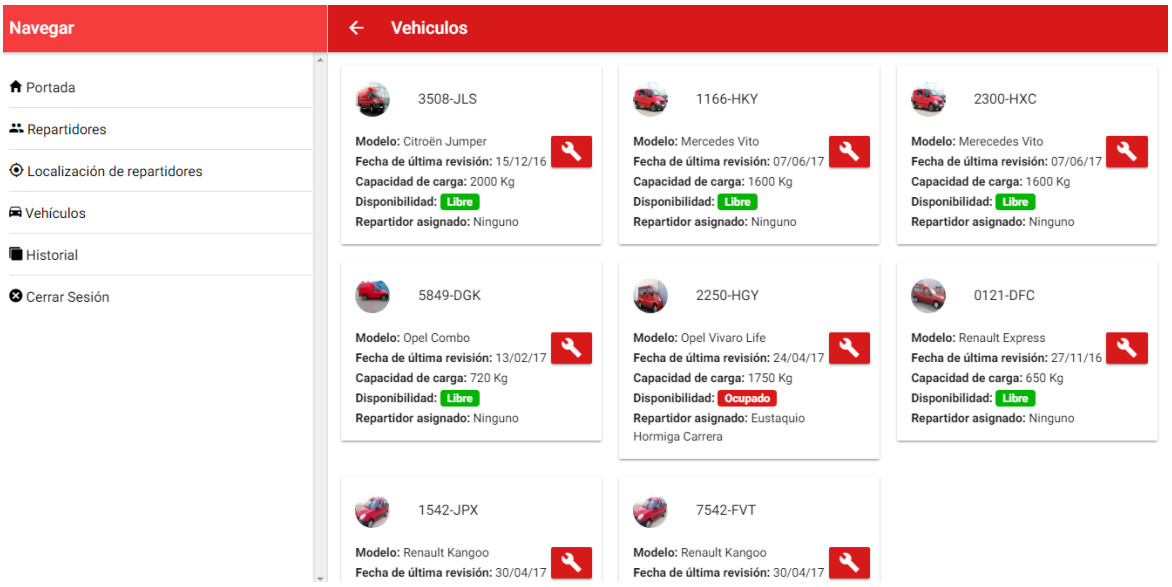

Ilustración 11: Página de vehículos

La página “Historial” nos muestra una lista en forma de tabla de todos los paquetes que han sido entregados, ordenada por fecha de entrega, tal y como podemos ver en la *Ilustración 12*. Como en la portada, para acceder a la información de cada paquete sólo tenemos que hacer click en alguno de ellos. El buscador también funciona como los de la portada.

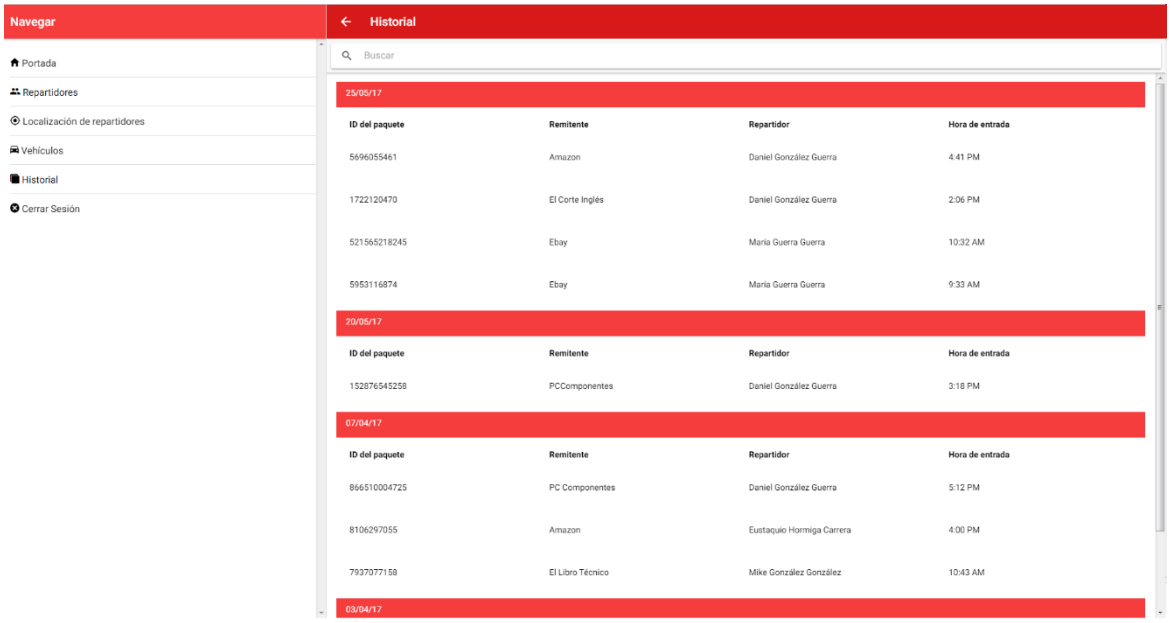

Ilustración 12: Página del historial

La última opción del menú lateral nos permite cerrar sesión en la aplicación y volver a la página de autenticación.

ANEXO II: Desarrollo del código

LoginPage

El primer paso para construir este prototipo consiste en crear una forma de iniciar sesión en la aplicación. En el código HTML se escribirá un formulario de dos campos y un botón (*Figura 1*).

login.page.html

```
<ion-content padding class="login-page" scroll="false">
  <img class="icon" src="../../assets/icon/login-icon.png" />
  <div class="row form-box">
    <form [formGroup]="loginForm" (submit)="loginUser()">
      <ion-item>
        <ion-input formControlName="email" type="email" placeholder="Correo electrónico"></ion-input>
      </ion-item>
      <ion-item>
        <ion-input formControlName="password" type="password" placeholder="Contraseña"></ion-input>
      </ion-item>
      <button ion-button class="login-button" type="submit">Acceder</button>
    </form>
  </div>
</ion-content>
```

Figura 1: Código HTML del formulario de autenticación

Y posteriormente la lógica de control de dicho formulario, que se detalla en el fichero "login.page.ts" y se muestra en la *Figura 2*. Es preciso destacar que se desactiva el menú lateral, para impedir el acceso a la aplicación sin iniciar sesión. Como todo método de autenticación, es preciso controlar la validez del correo electrónico introducido y los requisitos de la contraseña. Una vez introducidos los datos correctamente, y mientras se encuentre la sesión iniciada, la página que se encuentre en la parte superior de la pila, es decir, la raíz de la aplicación será HomePage, que se tratará a continuación.

login.page.ts

```
export class LoginPage {

  public loginForm: any;
  public loading: any;

  constructor(public navCtrl: NavController, public authData: AuthData,
    public formBuilder: FormBuilder, public alertCtrl: AlertController,
    public loadingCtrl: LoadingController, public menuController: MenuController) {

      this.menuController.enable(false);

      this.loginForm = formBuilder.group({
        email: ['', Validators.compose([Validators.required, EmailValidator.isValid])],
        password: ['', Validators.compose([Validators.minLength(6), Validators.required])]
      });
    }

    loginUser(){
      if (!this.loginForm.valid){…
      } else {
        if(this.loginForm.value.email != "admin@administrador.es"){…
        }else{
            this.authData.loginUser(this.loginForm.value.email, this.loginForm.value.password)
          .then( authData => {
            this.navCtrl.insert(0,HomePage);
            this.navCtrl.popToRoot();
          }, error => {
              let alert = this.alertCtrl.create({
                message: "La contraseña es incorrecta",
                buttons: [
                  {
                    text: "Ok",
                    role: 'cancel'
                  }
                ]
              });
              alert.present();
          });
```

Figura 2: Fragmento de código TypeScript para el control de inicio de sesión

HomePage

Una vez iniciada la sesión, aparece la página principal donde se gestionan los paquetes. La encabeza un mapa, implementado con Angular Google Maps, en el cual se muestran los paquetes que están sin asignar y los que ya han sido asignados a un repartidor. Para diferenciarlos se usa la etiqueta HTML "[iconUrl]" y, mediante una función, se elige un icono de marcador u otro en función del estado del paquete. Este mapa es inicializado en "home.page.ts" nada más cargar la página y es el HTML el encargado de mostrarlo.

home.page.html

```
11      <ion-toolbar no-border-top>
12          <ion-segment secondary [(ngModel)]="orderFilter">
13              <ion-segment-button value="notAssigned">
14                  Sin Asignar
15              </ion-segment-button>
16              <ion-segment-button value="assigned">
17                  Asignados
18              </ion-segment-button>
19              <ion-segment-button value="error">
20                  Incidencias
21              </ion-segment-button>
22          </ion-segment>
23      </ion-toolbar>
```

Figura 3: Código HTML para la creación de segmentos

A continuación, la página se divide en tres segmentos (*Figura 3*) mediante etiquetas HTML "<ion-segment>". Estos segmentos se controlan desde el fichero "home.page.html" como un selector, tal y como se observa en la *Figura 4*. Lo que quiere decir que, cuando se selecciona algún segmento, el HTML ignora las partes de los otros segmentos y sólo es ejecutada la correspondiente al segmento seleccionado. Por defecto, se ejecuta la parte del segmento "Sin Asignar".

home.page.html

```
39  <div [ngSwitch]="orderFilter" >
40      <ion-grid *ngSwitchCase="'notAssigned'">
41          <ion-toolbar >
42              <ion-searchbar placeholder="Buscar" [(ngModel)]="queryText" (ionInput)="search()"></ion-searchbar>
43          </ion-toolbar>
44          <ion-row >
45              <ng-container *ngFor="let deliverers of nAssignedOrders">
46                  <ion-col col-12 col-sm-6 col-md-6 col-lg-4 col-xl-4 *ngIf="deliverers.length != 0">
47                      <ion-card>
48                          <ion-item-divider color="secondary">{{deliverers.codPos}}</ion-item-divider>
49                          <ion-list >
50                              <ion-item-group>
51                                  <button *ngFor="let order of deliverers.pedido" ion-item icon-right outline (click)="goToOrder($event, order)">
52                                      <ion-row>
53                                          <ion-col col-10>
54                                              <h1>{{order.idPaquete}}</h1>
55                                              <h2><b>Remitente:</b> {{order.remitente}}</h2>
56                                              <h3><b>Dirección:</b> {{order.direccion}}</h3>
57                                              <p><b>Fech. Entrada:</b> {{order.fechaEntradaAlmacen | date:'dd/MM/yy'}} -
58                                               {{order.fechaEntradaAlmacen | date:'shortTime'}}</p>
59                                          </ion-col>
60                                          <ion-col col-2>
61                                              <br>
62                                              <ion-icon *ngIf="order.urgente" name="alert"></ion-icon>
63                                          </ion-col>
64                                      </ion-row>
65                                  </button>
66                              </ion-item-group>
67                          </ion-list>
68                      </ion-card>
69                  </ion-col>
70              </ng-container>
71          </ion-row>
72      </ion-grid>
```

Figura 4: Código HTML del primer segmento, "SIN ASIGNAR"

Cada segmento muestra una lista diferente de paquetes, a partir de tres filtrados distintos según el estado deseado del paquete en el fichero “home.page.ts” (*Figura 5*). El HTML muestra los objetos obtenidos de ese filtrado. Además de filtrar, las listas son mapeadas para ordenar los paquetes por código postal y dirección en el caso de los pedidos no asignados y por repartidor en caso de los pedidos asignados. Cada paquete posee una función “click” que redirecciona a OrderPage, menos aquellos mostrados en el segmento “Incidencias”.

home.page.ts

```
47  ionViewDidLoad(){
48
49      let loader = this.loadingController.create ({···
52      });
53
54      loader.present().then(() => {
55          this.angularFire.database.list('/pedidos').subscribe(data => {
56              this.ordersData = _.chain(data)
57                                 .filter(o => o.estado === "En el almacén")
58                                 .groupBy('codigoPostal')
59                                 .toPairs()
60                                 .map(item => _.zipObject(['codPos', 'pedido'], item))
61                                 .value();
62
63              this.nAssignedOrdersData =_.chain(this.ordersData)
64                                         .orderBy('direccion')
65                                         .value();
66
67              this.nAssignedOrders = this.nAssignedOrdersData;
68
69              this.allOrders = _.chain(data)
70                                .filter(o => o.estado === "En el almacén" || o.estado === "Asignado" ||
71                                             o.estado === "En reparto" || o.estado === "Siguiente en entrega" )
72                                .value();
73
74              this.ordersError = _.chain(data)
75                                  .filter(o => o.estado === "Incidencia registrada")
76                                  .value();
77
78              this.assignedOrdersData = _.chain(data)
79                                         .filter(o => o.estado === "Asignado")
80                                         .groupBy('repartidor')
81                                         .toPairs()
82                                         .map(item => _.zipObject(['repart', 'pedido'], item))
83                                         .value();
84
85              this.assignedOrders = this.assignedOrdersData;
86
87              loader.dismiss();
88          });
```

Figura 5: Código TypeScript con las diferentes listas de los segmentos

Los dos primeros segmentos poseen un buscador, el cual es controlado por una función en el fichero “home.page.ts” que permite varios campos de filtrado, como dirección, id del paquete o remitente. Al tratarse de dos listas diferentes (una contiene los paquetes asignados y otra los paquetes sin asignar) en los que se debe filtrar, es preciso hacer dos funciones de búsqueda diferentes.

Finalmente, por primera vez se observa el método que permite desplazarse a otras páginas, en concreto el que redirige a la página de un pedido. Este método se encuentra en la mayoría de las páginas implementadas.

OrderPage

En la primera parte de la página se muestra la información del paquete enviado por parámetro de navegación, además de un botón que redirecciona a MapPage (*Figura 7)*. La segunda parte de la página muestra también otros datos del mismo paquete y un botón para añadir una incidencia, siempre que el estado del paquete sea “En el almacén”, es decir, sin repartidor asignado. Este botón abrirá una pequeña ventana para redactar y guardar la incidencia que está controlada por el fichero “order.page.ts” mediante un AlertController (*Figura 6*). Una vez guardada la incidencia, la librería ToastController mostrará un mensaje de confirmación de la operación.

order.page.ts

```
120    addComment(){
121        let prompt = this.alertController.create({
122            title: 'Incidencia',
123            message: "Describe aquí la incidencia del paquete " + this.order.idPaquete,
124            inputs: [{
125                type: 'text',
126                name: 'incidencia'
127            }],
128            buttons: [
129                {
130                    text: 'Cancelar'
131                },
132                {
133                    text: 'Añadir',
134                    handler: data =>{
135                        this.orders.update(this.order.$key, {observaciones: data.incidencia, estado: "Incidencia registrada"});
136                        this.nav.popToRoot();
137                        let toast = this.toastController.create({
138                            message: "Se ha regsitrado la incidencia",
139                            duration: 4000,
140                            position: 'bottom'
141                        });
142                        toast.present();
143                    }
144                }
145            ]
146        });
147        prompt.present();
148    }
```

Figura 6: Código TypeScript de la función que añade una incidencia

home.page.ts

```
43      goToOrder($event, order){
44          this.nav.push(OrderPage, order);
45      }
46
```

Figura 7: Función que redirecciona a otra página

Por último, la tercera parte consiste en una tarjeta, “<ion-card>”, que mostrará diferentes funcionalidades según el estado del paquete mediante la función “ngIf” (*Figura 8*), la cual permite que se muestre algo siempre que se cumpla la condición deseada:

order.page.html

```
<ion-card *ngIf="!order.repartidor">
    <ion-card-content>
        <ion-slides pager>
            <ion-slide *ngFor="let deliveryMan of deliveryMen" (click)="assignDeliveryMan($event, deliveryMan)" >
                <ion-item class="avatar">
                    <ion-avatar>
                        <img src={{deliveryMan.imagen}}>
                    </ion-avatar>
                </ion-item>
                <p><b>Nombre:</b> {{deliveryMan.nombre}}</p>
                <p><b>Número de paquetes asignados:</b> {{deliveryMan.numPedidos}}</p>
                <p><b>Disponibilidad:</b> <ion-badge [color]="getCorrectColor(deliveryMan)">{{deliveryMan.disponibilidad}}</ion-badge></p>
                <p *ngIf="deliveryMan.coche"><b>Vehículo asignado:</b> {{deliveryMan.coche}}</p>
                <p *ngIf="!deliveryMan.coche"><b>Vehículo asignado:</b> Ninguno</p>
                <br>
            </ion-slide>
        </ion-slides>
    </ion-card-content>
</ion-card>

<button *ngIf="order.repartidor && !order.fechaEntrega" ion-button full (click)="removeDeliveryMan()" >
    Desasignar repartidor
</button>
```

Figura 8: Fragmento de código HTML que controla la tarjeta que debe ser mostrada

- Si el paquete está asignado a un repartidor, la tarjeta mostrará un botón para desasignarlo.

- Si aún no tiene asignado uno, mostrará un slider con los repartidores ordenados de forma ascendente en función del número de paquetes que ya tengan asignados. Este slider será controlado únicamente por el HTML a excepción de la obtención de los objetos del slider que se lleva a cabo en el TypeScript.

order.page.ts

```
assignDeliveryMan($event, deliveryMan){
    let prompt = this.alertController.create({
        title: 'Asignar',
        message: "¿Quieres asisgar este pedido al repartidor " + deliveryMan.nombre + "?",
        buttons: [
            {
                text: 'Cancelar'
            },
            {
                text: 'Si',
                handler: data =>{
                    this.orders.update(this.order.$key, {repartidor: deliveryMan.nombre, idRepartidor: deliveryMan.$key, estado: "Asignado"});
                    this.employees.update(deliveryMan.$key, {numPedidos: deliveryMan.numPedidos+1});
                    this.nav.popToRoot();

                    let toast = this.toastController.create({
                        message: "Se ha asignado el paquete " + this.order.idPaquete + " al repartidor " + deliveryMan.nombre,
                        duration: 4000,
                        position: 'bottom'
                    });
                    toast.present();
                }
            }
        ]
    });
    prompt.present();
}

removeDeliveryMan(){
    this.orders.update(this.order.$key, {repartidor: "", estado: "En el almacén", idRepartidor: ""});
    var numOrders = 0;
    var deliveryMan = this.angularFire.database.object(`repartidores/${this.order.idRepartidor}`).subscribe((deliveryMan) =>
            numOrders = deliveryMan.numPedidos-1
    );
    this.employees.update(this.order.idRepartidor, {numPedidos: numOrders})
    deliveryMan.unsubscribe();
    this.nav.popToRoot();

    let toast = this.toastController.create({
        message: "Se ha desasignado el paquete " + this.order.idPaquete + " del repartidor " + this.order.repartidor,
        duration: 4000,
        position: 'bottom'
    });
    toast.present();
}
```

Figura 9: Código TypeScript de las funciones para asignar y desasignar un repartidor

Al pulsar cualquiera de las dos tarjetas se abrirá una ventana de confirmación mediante un AlertController y un mensaje con un ToastController, como se muestra en la *Figura 9*.

Dado el caso de los paquetes entregados, la tarjeta no mostrará nada y, además, se facilitará información adicional sobre la entrega en la que se ha llamado la segunda parte de la página, como la firma del cliente, la fecha, hora y lugar exactos de la entrega, etc. (*Figura 10*).

order.page.html

```
        <ng-container *ngIf="order.estado === 'Entregado'">
            <ion-row>
                <ion-col>
                    <ion-item-divider color="secondary" >Información de la entrega</ion-item-divider>
                    <ion-row>
                        <ion-col>
                            <p><b>Hora de entrega:</b> {{order.fechaEntrega}}</p>
                            <p><b>Observaciones:</b> {{order.observaciones}}</p>
                        </ion-col>
                        <ion-col>
                            <ion-row>
                                <ion-col col-3>
                                    <p><b>Firma del cliente:</b></p>
                                </ion-col>
                                <ion-col col-9>
                                    <img class="signature" src="{{order.firma}}" *ngIf="order.firma" />
                                </ion-col>
                            </ion-row>
                        </ion-col>
                    </ion-row>
                </ion-col>
            </ion-row>
        </ng-container>
    </ion-list>
```

Figura 10: Código HTML que muestra información de paquetes entregados

DeliveryMenPage

En esta página se mostrará un listado de los repartidores, con pequeños datos como su nombre, imagen, número de paquetes asignados, vehículo asignado y siguiente dirección de entrega. En primer lugar, se escribe el código HTML para mostrar los repartidores, los cuales se mostrarán en tarjetas repartidas en columnas (*Figura 11*). En cada tarjeta se incluye la información de cada repartidor y un botón con un color variable en función de si dicho repartidor tiene un vehículo asignado o no (con la posibilidad de hacer click para desasignar el vehículo).

delivery-men.page.html

```
<ion-content>
    <ion-grid>
        <ion-row>
            <ion-col col-12 col-sm-6 col-md-6 col-lg-4 col-xl-4 *ngFor="let deliveryMan of deliveryMen">
                <ion-card (click)="goToDeliverer($event, deliveryMan)">
                    <ion-item>
                        <ion-avatar item-left>
                            <img src={{deliveryMan.imagen}}>
                        </ion-avatar>
                        <ion-card-header>
                            <b>{{deliveryMan.nombre}}</b>
                        </ion-card-header>
                    </ion-item>
                    <ion-row>
                        <ion-col col-10>
                            <ion-card-content>
                                <p><b>DNI:</b> {{deliveryMan.dni}}</p>
                                <p><b>Número de paquetes asignados:</b> {{deliveryMan.numPedidos}}</p>
                                <p><b>Disponibilidad:</b> <ion-badge [color]="getCorrectColor(deliveryMan)">{{deliveryMan.disponibilidad}}</ion-badge></p>
                                <p *ngIf="deliveryMan.coche"><b>Vehículo asignado:</b> {{deliveryMan.coche}}</p>
                                <p *ngIf="!deliveryMan.coche"><b>Vehículo asignado:</b> Ninguno</p>
                                <p *ngIf="deliveryMan.sigDireccion"><b>Siguiente Dirección de reparto:</b> {{deliveryMan.sigDireccion}}</p>
                            </ion-card-content>
                        </ion-col>
                        <ion-col col-2>
                            <button ion-button color="verde" icon-only *ngIf="!deliveryMan.coche" >
                                <ion-icon name="car"></ion-icon>
                            </button>
                            <button ion-button icon-only (click)="removeCar($event, deliveryMan)" *ngIf="deliveryMan.coche" >
                                <ion-icon name="close-circle"></ion-icon>
                            </button>
                        </ion-col>
                    </ion-row>
                </ion-card>
            </ion-col>
        </ion-row>
    </ion-grid>

</ion-content>
```

Figura 11: Código HTML que muestra los repartidores

Para ello se han empleado las etiquetas <ion-grid>, <ion-col> y finalmente <ion-card>, que crearán una cuadrícula, la dividirán en columnas y la poblarán con tarjetas que contendrán la información del repartidor, tal y como puede verse a continuación:

Se pasa a la lógica de control, donde se ve en primer lugar el método "ionViewDidLoad" (*Figura 12*), que se ejecutará en el primer instante que la página se muestre. En dicho método se encuentra la operación de lectura de la base de datos, con su posterior filtrado y ordenación por el campo 'nombre', que se encargará de mostrar información sobre los repartidores en las tarjetas. Así mismo, se inician dos lecturas a la base de datos para mostrar los vehículos y obtener un listado sin filtrar de los repartidores.

delivery-men.page.ts

```
33    ionViewDidLoad(){
34        let loader = this.loadingController.create({
35            content: 'Cargando...',
36            spinner: 'bubbles'
37        });
38        loader.present().then(() => {
39            this.angularFire.database.list('/repartidores').subscribe(data => {
40                this.allMen =
41                    _.chain(data)
42                    .orderBy('nombre')
43                    .value();
44                this.deliveryMen = this.allMen;
45                this.vehiclesDatabase = this.angularFire.database.list
                  ('/coches');
46                this.deliveryMenDatabase = this.angularFire.database.list
                  ('/repartidores');
47                loader.dismiss();
48            });
49        });
50    }
```

Figura 12: Código TypeScript de la función "ionViewDidLoad"

A continuación, en la *Figura 13* se encuentran tres pequeños métodos, siendo el primero de ellos para calcular la disponibilidad y mostrar un color rojo si estuviera ocupado o verde en caso de estar libre. Los dos siguientes son empleados para desplazarse a otra página. La instrucción "push" aplicada a un elemento de control de navegación permite desplegar una nueva página. El primer método se corresponde con la función "click" de la tarjeta, mientras que el segundo corresponde a un botón que mostrará un mapa con la localización de los repartidores.

delivery-men.page.ts

```
53    getCorrectColor(deliveryMan){
54        return deliveryMan.disponibilidad === "En ruta" ? 'primary' : 'verde';
55    }
56
57
58    goToDeliverer($event, deliveryMan){
59        this.nav.push(DelivererHomePage, deliveryMan);
60    }
61
62    goDelivererLocation(){
63        this.nav.push(DelivererLocationPage);
64    }
```

Figura 13: Funciones para cambiar el color del estado y redireccionar a otras páginas

Finalmente, se encuentra el método "removeCar" (*Figura 14*), que se encarga de desasignar el vehículo que tuviera asignado el repartidor (y a su vez, desasignar el repartidor del vehículo), además de actualizar su información a "Libre". Al terminar las operaciones de actualización de la base de datos, se muestra un mensaje de confirmación.

delivery-men.page.ts

```
66    removeCar($event, deliverer){
67        this.angularFire.database.list('/coches').subscribe(data =>{
68            this.carsData = _.chain(data)
69                             .filter(c => c.matricula === deliverer.coche)
70                             .value();
71            this.vehiclesDatabase.update(this.carsData[0].$key, {repartidor: "", disponibilidad: "Libre"});
72        });
73
74        this.deliveryMenDatabase.update(deliverer.$key, {coche: ""});
75
76        let toast = this.toastController.create({
77            message: "Se ha desasignado el vehículo al repartidor",
78            duration: 4000,
79            position: 'bottom'
80        });
81
82        toast.present();
83    }
```

Figura 14: Función para desasignar un vehículo

DelivererHomePage

La siguiente página es en realidad un marco para tres páginas distintas que se asocian a las tres pestañas que contiene dicho marco (*Figura 15*), mostrando una página u otra en función de la pestaña seleccionada. Las páginas que contiene son *DelivererPage*, *DelivererPacksPage* y *DelivererPackFinishPage*, que detallan toda la información necesaria sobre el repartidor elegido.

delivererHome.page.ts

```
export class DelivererHomePage {

    deliverer: any;
    delivererTab = DelivererPage;
    delivererPacksTab = DelivererPacksPage;
    delivererPackFinishTab = DelivererPackFinishPage;

    constructor(private nav: NavController,
                private navParams: NavParams){
        this.deliverer = this.navParams.data;
    }
}
```

delivererHome.page.html

```
<ion-header>
    <ion-navbar color="primary">
        <ion-title>{{deliverer.nombre}}</ion-title>
    </ion-navbar>
</ion-header>

<ion-tabs>
    <ion-tab tabIcon="contact" tabTitle="Info" [root]="delivererTab"
    [rootParams]="deliverer"></ion-tab>
    <ion-tab tabIcon="clipboard" tabTitle="Asignados" [root]
    ="delivererPacksTab" [rootParams]="deliverer"></ion-tab>
    <ion-tab tabIcon="archive" tabTitle="Entregados" [root]
    ="delivererPackFinishTab" [rootParams]="deliverer"></ion-tab>
</ion-tabs>
```

Figura 15: Código HTML y TypeScript para la integración de pestañas

DelivererPage

La primera pestaña de las comentadas en el apartado anterior se corresponde a una página dedicada a mostrar información sobre el repartidor seleccionado. En primer lugar, el código HTML empleado, el cual se muestra en la *Figura 16*, mostrará una sección con una serie de campos que contienen información relevante y la imagen del repartidor. La siguiente sección contiene los paquetes seleccionados por el repartidor para entregar y un mapa con la dirección de entrega del siguiente pedido que se ha seleccionado. Finalmente, se encuentra la sección dedicada a asignar un vehículo al repartidor, pudiendo escoger, con un simple click, de entre una lista de vehículos que no se encuentren en uso. Así mismo, en caso de tener un vehículo asignado, esta última sección no se mostrará, conteniendo en su lugar un botón que permita desasignar dicho vehículo.

deliverer.page.html

```
<ng-container *ngIf="deliverer.coche; else elseBlock">
    <ion-item-divider color="secondary" >Paquetes seleccionados</ion-item-divider>
    <ion-list >
        <ion-item *ngFor="let order of orders"  (click)="goToOrder($event, order)" [class.next-order]="order.estado == 'Siguiente en entrega'" >
            <p>{{order.idPaquete}}</p>
            <br>
            <p>{{order.direccion}}</p>
        </ion-item>
    </ion-list>
    <ion-item-divider color="secondary" >Ruta del repartidor</ion-item-divider>
    <ion-card class="deliveryMap-page" *ngIf="deliverer.horaCapturaGPS">
        <agm-map id="map" [latitude]="map.lat" [longitude]="map.lng" [zoom]="map.zoom" >
            <agm-marker [latitude]="map.lat" [longitude]="map.lng">
                <agm-info-window>
                    <strong>{{map.markerLabel}}</strong>
                </agm-info-window>
            </agm-marker>

            <agm-marker *ngIf="nextOrder" [latitude]="nextOrder.latitud" [longitude]="nextOrder.longitud" [iconUrl]="'../../assets/marker-icons/marker_orange.png'">
                <agm-info-window>
                    <strong>{{nextOrder.direccion}}</strong>
                </agm-info-window>
            </agm-marker>
        </agm-map>
    </ion-card>
    <br>
    <button  ion-button full (click)="removeCar()">
        Desasignar vehículo
    </button>
</ng-container>

<ng-template #elseBlock>
    <ion-item-divider color="secondary" >Vehículos disponibles</ion-item-divider>
    <ion-row>
        <ion-list *ngFor="let vehicle of vehicles">
            <ion-col>
                <ion-item (click)="addCar($event, vehicle)" >
                    <ion-avatar item-left>
                        <img src={{vehicle.imagen}}>
                    </ion-avatar>
                    <h2>{{vehicle.matricula}}</h2>
                    <p>{{vehicle.modelo}}</p>
                </ion-item>
            </ion-col>
        </ion-list>
    </ion-row>
</ng-template>
</ion-content>
```

Figura 16: Código HTML de la pestaña "Info"

En el código TypeScript se encuentra la lógica de control del HTML anteriormente descrito, donde destaca el método "ionViewDidLoad" (*Figura 17*), encargado de obtener y filtrar los datos de pedidos que el repartidor haya seleccionado para entregar. En dicho método se encuentra además la creación del mapa (que se posiciona sobre la localización geográfica del repartidor), la obtención de un listado de vehículos disponibles y la inicialización de dos instancias de datos sobre vehículos y repartidores, de forma que posteriormente se puede editar información de la base de datos.

deliverer.page.ts

```
ionViewDidLoad(){
    this.deliverer = this.navParams.data;
    let loader = this.loadingController.create({…
    });

    loader.present().then(() => {
        this.angularFire.database.list('/pedidos').subscribe(data => {
            this.ordersData = _.chain(data)
                               .filter(o => o.idRepartidor === this.deliverer.$key && o.estado == "En reparto" || o.estado == "Siguiente en entrega")
                               .value();
            this.orders = this.ordersData;
            for (var i = 0; i< this.ordersData.length; i++){…
            }
        });

        this.map = {…
        this.angularFire.database.list('/coches').subscribe(data => {
            this.vehiclesData = _.chain(data)
                                 .filter(v => v.disponibilidad === "Libre")
                                 .value();

            this.vehicles = this.vehiclesData;
            this.vehiclesDatabase = this.angularFire.database.list('/coches');
            this.deliveryMen = this.angularFire.database.list('/repartidores');

            loader.dismiss();
        });
    });
}
```

Figura 17: Función "ionViewDidLoad"

A continuación, como se ve en la *Figura 18*, el método "addCar" es empleado en el proceso de asignación de vehículos a repartidores. Su lógica se basa en la creación de una ventana emergente que se mostrará al usuario cuando pulse sobre el vehículo a asignar, en cuyo momento se pedirá confirmación sobre dicha asignación. En caso de responder con "Si", se actualizará la instancia de vehículos (el vehículo seleccionado tendrá un repartidor asignado y su estado pasa a "Ocupado") y la instancia de repartidores (la información del repartidor reflejará que tiene un coche asignado) de la base de datos. Terminadas estas operaciones, se mostrará un mensaje que indicará que las tareas se han ejecutado correctamente.

El método "removeCar" es empleado para desasignar el vehículo que tenga asignado un repartidor. Para ello, se busca el vehículo que coincida con el asignado al repartidor y se actualiza su información para eliminar dicha asignación. Del mismo modo, en la instancia de la base de datos correspondiente al repartidor, se elimina también la información del vehículo asignado. Al finalizar este proceso se muestra un mensaje informativo.

deliverer.page.ts

```
addCar($event, vehicle){
    let prompt = this.alertController.create({
        title: 'Asignar',
        message: "¿Quieres añadir el vehículo con matrícula " +
        vehicle.matricula + " a este repartidor? ",
        buttons: [
            {
                text: 'Cancelar'
            },
            {
                text: 'Si',
                handler: data =>{
                    this.vehiclesDatabase.update(vehicle.$key, {repartidor:
                    this.deliverer.nombre,  disponibilidad: "Ocupado"});
                    this.deliveryMen.update(this.deliverer.$key, {coche:
                    vehicle.matricula});

                    let toast = this.toastController.create({···
                    });

                    toast.present();
                }
            }
        ]
    });
    prompt.present();
}

removeCar(){
    this.angularFire.database.list('/coches').subscribe(data =>{
        this.carsData = _.chain(data)
                        .filter(c => c.matricula === this.deliverer.coche)
                        .value();

        this.car = this.carsData;
        this.vehiclesDatabase.update(this.car[0].$key, {repartidor: "",
        disponibilidad: "Libre"});

    });

    this.deliveryMen.update(this.deliverer.$key, {coche: ""});

    let toast = this.toastController.create({···
    });

    toast.present();
}
```

Figura 18: Funciones para asignar y desasignar vehículos

Finalmente, se encuentran dos pequeños métodos que se han descrito en secciones anteriores y que son empleados para desplazarse a otra página o para obtener el color correcto del marcador del mapa en función del estado del repartidor.

DelivererPacksPage

La segunda pestaña dedicada a los repartidores contiene un listado de los pedidos que tenga asignado el repartidor seleccionado. Como sucede en la página principal *HomePage*, los pedidos se generan en listas dentro de tarjetas, que a su vez tienen cabeceras en función del código postal de la dirección de entrega del pedido. Desde el momento que exista un pedido asignado al repartidor, se mostrará un mapa que contendrá la localización de entrega de los pedidos.

Pasando al código TypeScript, se encuentra el método "ionViewDidLoad", que en esta ocasión se encarga de recoger los datos pasados como parámetros (en concreto el repartidor seleccionado), crear el mapa y finalmente, obtener de la base de datos una lista de pedidos

asignados a dicho repartidor, ordenarlos por código postal y posteriormente por dirección de entrega (*Figura 19*), para que el HTML pueda mostrarlo correctamente.

delivererPacks.page.ts

```
27    ionViewDidLoad(){
28        this.deliverer = this.navParams.data;
29
30        this.map = {···
36        this.angularFire.database.list('/pedidos').subscribe(data => {
37
38            this.ordersData = _.chain(data)
39                               .filter(o => o.idRepartidor === this.deliverer.$key && o.fechaEntrega === "")
40                               .groupBy('codigoPostal')
41                               .toPairs()
42                               .map(item => _.zipObject(['codPos', 'pedido'], item))
43                               .value();
44
45            this.orders = _.chain(this.ordersData)
46                                .orderBy('direccion')
47                                .value();
48        });
49    }
```

Figura 19: Código TypeScript de la lista de paquetes asignados a un repartidor

DelivererPackFinishPage

La tercera y última pestaña está dedicada a los pedidos que ese repartidor ha entregado, por lo que es idéntica tanto en su estructura HTML como TypeScript a la página *DelivererPacksPage*, debido a que tienen el mismo diseño y la lógica de control solo difiere en la instrucción de filtrado. En este caso, se filtrarán los pedidos por aquellos que, además de estar asignados al repartidor seleccionado, hayan sido entregados, para lo que se utilizará la expresión ".filter(o => o.idRepartidor === this.deliverer.$key && o.fechaEntrega != "")".

DelivererLocationPage

Esta página es la encargada de mostrar un mapa con la ubicación en tiempo real de todos los repartidores con un coche asignado, tanto los que se encuentra en ruta de reparto, con su propio marcador, como los que se encuentran de camino al almacén. Para ello, se usa la etiqueta HTML "[iconUrl]" junto a una función para determinar qué marcador se debe aplicar en cada caso (*Figura 20*).

delivererLocation.page.html

```
7   <ion-content class="location-page">
8       <agm-map id="map" [latitude]="map.lat" [longitude]="map.lng" [zoom]="map.zoom" >
9           <ng-container *ngFor="let deliveryMan of deliverer">
10              <agm-marker [latitude]="deliveryMan.latitud" [longitude]="deliveryMan.longitud" [iconUrl]="getCorrectColor(deliveryMan)">
11                  <agm-info-window >
12                      <strong (click)="goToDeliverer($event, deliveryMan)">{{deliveryMan.nombre}}</strong>
13                  </agm-info-window>
14              </agm-marker>
15          </ng-container>
16      </agm-map>
17  </ion-content>
```

Figura 20: Código HTML que implementa el mapa

La inicialización de dicho mapa proviene del código en TypeScript, que, al iniciarse la página, el método "ionViewDidLoad" entra en acción para crear la lógica del mapa, situarlo sobre unas coordenadas establecidas y realizar una búsqueda en la base de datos para obtener los datos de los repartidores que tengan un coche asignado y hayan transmitido datos de su localización a través de su aplicación (*Figura 21*). Finalmente, gracias a las estructuras propias de la librería Angular Google

Maps, se crea la visualización del mapa y se generan los marcadores apropiados. Es importante ajustar correctamente el fichero "delivererLocation.page.scss" para que el mapa ocupe toda la página.

delivererLocation.page.ts

```
ionViewDidLoad(){
    this.map = {
        lat: 27.942246703329612,
        lng: -15.598526000976562,
        zoom: 10
    };

this.angularFire.database.list('/repartidores').subscribe(data => {
        this.delivererData = _.chain(data)
                              .filter(o => o.coche != "" && o.horaCapturaGPS !="")
                              .value();
        this.deliverer = this.delivererData;
    });
}

getCorrectColor(deliveryMan){
    return deliveryMan.disponibilidad === "En ruta" ?
        '../../assets/marker-icons/marker_red.png' :
        '../../assets/marker-icons/marker_green.png';
}
```

Figura 21: Código de inicialización del mapa

VehiclesPage

Esta página ofrece la lista de vehículos de los que dispone la empresa. A través del fichero "vehicles.page.ts" se obtienen los vehículos de la tabla del Firebase y se muestran con el HTML a modo de tarjetas, "<ion-card>". Gracias a la etiqueta "<ion-avatar>", cada tarjeta muestra una imagen en miniatura del vehículo junto a su número de matrícula.

A la derecha de cada tarjeta se observa un botón con el icono de una herramienta donde, mediante la función "newRevision" (*Figura 22*), actualiza la fecha de la última revisión superada por el vehículo enviado como parámetro. Acto seguido aparece una ventana de confirmación creada por el AlertController y un mensaje por un ToastController después de haber confirmado la operación.

El cambio de la fecha de última revisión de un vehículo se obtiene a partir de la librería "Moment", la cual recoge la fecha y la hora del sistema en el momento en el que es llamada y actualiza el valor en la base de datos.

vehicles.page.ts

```
newRevision(car){

    let prompt = this.alertController.create({
        title: 'Fecha de última revisión',
        message: "¿Quieres actualizar la fecha de la última revisión de este vehículo?",
        buttons: [
            {
                text: 'Cancelar'
            },
            {
                text: 'Si',
                handler: data =>{
                    this.vehiclesDatabase.update(car.$key, {ultimaRevision: moment().format()});

                    let toast = this.toastController.create({
                        message: "Se ha actualizado la fecha de la última revisión del vehículo " + car.matricula,
                        duration: 4000,
                        position: 'bottom'
                    });

                    toast.present();
                }
            }
        ]
    });
    prompt.present();
}
```

Figura 22: Función para actualizar la última fecha de revisión de un vehículo

RegistryPage

En esta página se muestra una lista de paquetes entregados ordenada por el campo "fechaEntrega" de la tabla "Pedidos". Para ello se filtra la tabla de Firebase para obtener sólo los paquetes con estado entregado y se realiza un mapeo para la ordenación que separa los objetos obtenidos en el filtrado por el campo deseado, en este caso el de la fecha en el que han sido entregados.

El fichero "registry.page.html" (*Figura 23*) muestra estos paquetes colocándolos en tablas mediante una cuadrícula (etiquetas <ion-row> e <ion-col>) que consiguen una mejor visualización de la información.

registry.page.html

```
<ion-content padding>
    <ion-list *ngFor="let fechaEntrega of orders" >
        <ion-item-group *ngIf="fechaEntrega.date" >
                <ion-item-divider color="secondary">
                    <h3>{{fechaEntrega.date | date:'dd/MM/yy'}}</h3>
                </ion-item-divider>

            <ion-row class="text">
                <ion-col class="col-3" >
                    <p><b>ID del paquete</b></p>
                </ion-col>

                <ion-col class="col-3" >
                    <p><b>Remitente</b></p>
                </ion-col>

                <ion-col class="col-3" >
                    <p><b>Repartidor</b></p>
                </ion-col>

                <ion-col class="col-3" >
                    <p><b>Fecha de entrada</b></p>
                </ion-col>
            </ion-row>

            <ion-list *ngFor="let order of fechaEntrega.order" (click)="itemTapped($event, order)">
                <ion-row class="text">
                <ion-col class="col-3" >
                    <p>{{order.idPaquete}}</p>
                </ion-col>

                <ion-col class="col-3" >
                    <p>{{order.remitente}}</p>
                </ion-col>

                <ion-col class="col-3" >
                    <p>{{order.repartidor}}</p>
                </ion-col>

                <ion-col class="col-3" >
                    <p>{{order.fechaEntrega | date:'dd/MM/yy'}}
                        {{order.fechaEntrega | date:'shortTime'}}</p>
                </ion-col>
                </ion-row>

            </ion-list>
        </ion-item-group>
    </ion-list>
</ion-content>
```

Figura 23: Código HTML de la página Historial

Además se dispone de un buscador (Figura 24) que, mediante la función “search()”, cuyo código se muestra en la *Figura 25*, recoge el valor obtenido en el input y lo filtra por varios campos como repartidor, remitente, fecha de entrega, etc. De esta manera se muestran sólo los paquetes en los que alguna parte de su información coincida con las palabras filtradas.

registry.page.html

```
<ion-header>
    <ion-navbar color="primary">
        <ion-title>Historial</ion-title>
    </ion-navbar>

    <ion-toolbar >
        <ion-searchbar placeholder="Buscar" [(ngModel)]="queryText" (ionInput)="search()"></ion-searchbar>
    </ion-toolbar>
</ion-header>
```

Figura 24: Código HTML que implementa el buscador

```
    search(){
        let queryTextLower = this.queryText.toLowerCase();
        let filteredOrders = [];

        _.forEach(this.allDates, dat => {
            let orders = _.filter(dat.order, or => (<any>or).repartidor.toLowerCase()
            .includes(queryTextLower) || (<any>or).fechaEntrega.toLowerCase()
            .includes(queryTextLower) || (<any>or).remitente.toLowerCase()
            .includes(queryTextLower) || (<any>or).idPaquete.toString().includes(queryTextLower));
            if (orders.length) {
                filteredOrders.push({ date: dat.date, order: orders});
            }
        });

        this.orders = filteredOrders;
    }
```

Figura 25: Código TypeScript que controla el filtrado del buscador

Por último, los paquetes poseen un método “click” que permite redireccionar al usuario a OrderPage al pulsar sobre ellos.

MapPage

Esta página consta de un mapa en el que se muestra la dirección de entrega de un pedido, que, como se ha detallado en el apartado dedicado a la página de la localización de los repartidores, se trata de un procedimiento sencillo. El mapa es inicializado por el fichero “map.page.ts” al cargar la página, posicionándose sobre la localización de entrega del pedido seleccionado como parámetro y con un marcador con la dirección. Finalmente, el código HTML es el encargado de mostrarlo. Como se ha descrito anteriormente, el fichero “map.page.scss” juega un papel muy importante debido a que sin él, el mapa no se mostraría. En la *Figura 26* se muestran ambos códigos.

map.page.html

```
<ion-content class="map-page">
    <agm-map id="map" [latitude]="map.lat" [longitude]="map.lng" [zoom]="map.zoom" >
        <agm-marker [latitude]="map.lat" [longitude]="map.lng">
            <agm-info-window>
                <strong>{{map.markerLabel}}</strong>
            </agm-info-window>
        </agm-marker>
    </agm-map>
</ion-content>
```

map.page.scss

```
.map-page #map {
  width: 100%;
  height: 100%;
  transition: opacity 150ms ease-in
}
```

Figura 26: Código HTML y SCSS que implementan el mapa