UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

iMediador

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Joseph Morant Navarro

Tutor: María Carmen Penadés Gramage

Cotutor: Pedro Jorquera Hervás

2015/2016

A mis padres.

Por los valores que me han inculcado y por su apoyo incondicional.

Agradecimientos

A mi tutora, Maria Carmen Penadés Gramage, por haberme ayudado a realizar el presente Trabajo de Fin de Grado.

A mi cotutor, Pedro Jorquera Hervás, por ofrecerme la posibilidad de formar parte de Okode.

Al resto del equipo de Okode, por tratarme como un miembro más desde el primer momento.

A mis familiares y amigos, por su paciencia y el apoyo que me permitió seguir adelante.

Resumen

En el presente Trabajo de Fin de Grado se presenta la migración de una aplicación que permite al Colectivo de Mediadores de MAPFRE realizar una serie de funcionalidades útiles durante la jornada laboral, en concreto, el mantenimiento de listas de gestiones, la programación de planes de visita y la administración de contactos o vencimientos asociados a los mismos, ofreciendo la posibilidad de visualizar la información de localización de contactos y planes de visita sobre mapas interactivos donde se geolocaliza cada punto de interés. Como funcionalidades de valor añadido, se permite guardar la posición del vehículo, establecer una alarma para las zonas de estacionamiento regulado y la notificación de eventos relevantes a través de tecnología *push*.

Los datos con los que trabaja la aplicación se obtienen de un *backend* basado en una API REST (desarrollado por un tercero), aunque también existe la posibilidad de guardar cierta información de manera local para su posterior sincronización con éste.

Puesto que la aplicación nativa fue desarrollada para Android, iOS y Windows Phone en lenguajes distintos (Java, Objective-c/Swift y C# respectivamente) y para la nueva se utiliza un *framework* basado en JavaScript, no se ha podido reutilizar el código nativo, aunque en algunos casos ha servido como inspiración.

Para el desarrollo se ha utilizado una metodología ágil, siendo la duración del mismo de tres meses, con entregas tempranas y continuas de betas cada semana, así como liberación de versiones estables cada mes.

La aplicación se ha implementado haciendo uso de una herramienta que facilita el desarrollo multiplataforma: *Kony*.

Palabras clave: *Kony*, API REST, multiplataforma, gestiones, contactos, planes de visita, vencimientos, geolocalización, mapas interactivos, rutas, aparcamiento.

Abstract

This Degree Final Project deals about a migration of an application that allows to MAPFRE Mediators Collective to perform a set of useful features during the workday, specifically, managements list maintenance, visit plans schedule, and contacts or expirations administration, offering the ability to visualize contacts or visit plans location on interactive maps and geolocating each point of interest. As extra capabilities, it is possible to store the vehicle position, to set an alarm for payed parking zones and to notify about relevant events through push technology.

Used data is obtained from an RESTful API based backend (developed by third party) although it is also possible to store certain information locally and synchronize as soon as possible.

The native application was developed for Android, iOS and Windows Phone in different languages (Java, Objective-c/Swift and C# respectively) but for the new one, a JavaScript based framework is used. So, native code cannot be reused, but it has been inspiring.

The project has been developed using an agile methodology and it has lasted three months, with early and continuous beta deliveries every week and stable versions releases every month.

The application has been implemented using a tool that makes easier multiplatform development: *Kony*.

Keywords: *Kony*, RESTful API, multiplatform, managements, contacts, visit plans, expirations, geolocation, interactive maps, routes, parking.

Tabla de contenidos

1. Introducción 13
1.1. Motivación 13
1.2. Objetivos 14
1.2.1. Objetivo general 14
1.2.2. Objetivos particulares 15
1.3. Estructura del documento 15
2. Estado del arte 17
2.1. iMediador como aplicación nativa 17
2.1.1. Requisitos de la aplicación nativa 18
2.1.1.1. Requisitos funcionales 18
2.1.1.2. Requisitos no funcionales 20
2.1.2. Arquitectura de la aplicación nativa 23
2.1.3. Particularidades de cada plataforma 25
2.1.3.1. Desarrollo de la solución en iOS 25
2.1.3.2. Desarrollo de la solución en Android 25
2.1.3.3. Desarrollo de la solución en Windows Phone 25
2.2. Plataforma de Movilidad Kony 25
3. iMediador v2.0 27
3.1. Ámbito 27
3.1.1. Procesos de negocio afectados 27
3.1.2. Entidades y unidades involucradas en el proyecto 27
3.1.3. Usuarios a los que va dirigida la aplicación 28
3.1.4. Ámbito geográfico 28
3.2. Infraestructura 28
3.3. Entorno tecnológico 30
4. Modelado Conceptual 33
4.1. Modelo de Casos de Uso 33
4.2. Diagrama de Clases 39
4.3. Modelo de Navegación 40

5. Diseño del sistema 43
5.1. Arquitectura de la aplicación 43
5.1.1. Introducción 43
5.1.2. Diagrama de la arquitectura 44
5.2. Capa de servicios 48
5.2.1. Servicio de mapas y geolocalización 48
5.2.2. Modelo de comunicación con sistemas externos 49
5.3. Seguridad 49
6. Desarrollo de la solución 53
6.1. Introducción 53
6.2. Estructura del proyecto 54
6.3. Dependencias 59
6.3.1. Notificaciones push 59
6.3.2. Estadisticas 60
6.4. Gestión y configuración de la aplicación en remoto 60
6.4.1. Valores configurables desde MMobile 61
6.4.2. Funcionalidades configurables desde MMobile 61
6.5. Pruebas de *software* 61
6.6. Gestión de memoria y uso de CPU 62
6.6.1. Uso de CPU y memoria en Android 62
6.6.2. Uso de memoria y CPU en iOS 64
7. Conclusiones 65
8. Bibliografía 67
Apéndices 70
A. Manual de usuario 71

Índice de tablas

Tabla 1: CU01 – Mostrar listado de gestiones sin asignar 35
Tabla 2: CU02 – Mostrar detalle de una gestión sin asignar 35
Tabla 3: CU03 - Asignar una gestión 36
Tabla 4: CU04 - Ordenar gestiones manualmente 36
Tabla 5: CU05 - Ordenar gestiones cronológicamente 36
Tabla 6: CU06 - Ordenar gestiones por ruta óptima 37
Tabla 7: CU07 - Obtener dirección actual 37
Tabla 8: CU08 - Cerrar gestión 37
Tabla 9: CU09 – Buscar por teléfono 37
Tabla 10: CU10 - Buscar por email 38

Índice de ilustraciones

Ilustración 1: Arquitectura global de la app nativa (extraída de [8]) 17
Ilustración 2: Diagrama de la arquitectura de la app nativa (extraída de [7]) 24
Ilustración 3: Capas de la app nativa (extraída de [7]) 24
Ilustración 4: Gestiones sin asignar 34
Ilustración 5: Ordenación del plan de visitas 34
Ilustración 6: Geolocalización de direcciones 34
Ilustración 7: Cierre de gestiones 35
Ilustración 8: Ampliar criterios de búsqueda de contactos 35
Ilustración 9: Diagrama de clases de la aplicación 39
Ilustración 10: Modelo de navegación - Menú principal 40
Ilustración 11: Modelo de navegación - Contactos 40
Ilustración 12: Modelo de navegación - Gestiones 41
Ilustración 13: Modelo de navegación - Gestiones sin asignar 41
Ilustración 14: Modelo de navegación - Plan de visitas 41
Ilustración 15: Modelo de navegación - Parking 42
Ilustración 16: Diagrama de la arquitectura global (extraído de [1]) 44
Ilustración 17: Diagrama de la arquitectura 45
Ilustración 18: Código del controlador de 'búsqueda de clientes' 46
Ilustración 19: Modelo de la entidad Cliente 47
Ilustración 20: Código del viewModel de 'búsqueda de clientes' 48
Ilustración 21: Estructura general del proyecto 54
Ilustración 22: Formularios de la app 55
Ilustración 23: Ficheros de código fuente (1 de 2) 56
Ilustración 24: Ficheros de código fuente (2 de 2) 57
Ilustración 25: Activación de notificaciones 60
Ilustración 26: Gráfica de uso de CPU en Android 62
Ilustración 27: Gráfica de uso de CPU de ‘Contactos’ en Android 62
Ilustración 28: Gráfica de uso de memoria en Android 63
Ilustración 29: Gráfica de uso de memoria del ‘Plan de visitas’ en Android 63
Ilustración 30: Gráfica de uso de CPU en iOS 64
Ilustración 31: Gráfica de uso de memoria en iOS 64
Ilustración 32: Login 71
Ilustración 33: Menú principal 72
Ilustración 34: Gestiones sin asignar 72
Ilustración 35: Gestiones sin asignar - Listado de gestiones 73
Ilustración 36: Gestiones sin asigna - Detalle de gestión 74
Ilustración 37: Gestiones 74
Ilustración 38: Listado de gestiones 75

Ilustración 39: Detalle de gestión 76
Ilustración 40: Detalle de contacto 77
Ilustración 41: Contactos 77
Ilustración 42: Listado de contactos 78
Ilustración 43: Detalle de contacto 79
Ilustración 44: Plan de visitas 79
Ilustración 45: Detalle de gestión 80
Ilustración 46: Ubicación 81
Ilustración 47: Parking 81
Ilustración 48: Ajustes 82
Ilustración 49: Perfil 83
Ilustración 50: Modificar PIN 83

1. Introducción

1.1. Motivación

El desarrollo del presente Trabajo de Fin de Grado surge a raíz de la realización de prácticas en la empresa Okode. Durante la duración de las mismas, se presentó la oportunidad de realizar la migración de un proyecto existente: **iMediador**.

iMediador es una Aplicación Móvil Multiplataforma para el Colectivo de Mediadores de MAPFRE que permite implementar una serie de funcionalidades que típicamente realizan desde sus dispositivos móviles durante la jornada laboral, como puede ser el mantenimiento de listas de gestiones, la programación de planes de visita y la administración de contactos o vencimientos asociados a los mismos, ofreciendo la posibilidad de visualizar la información de localización de contactos y planes de visita sobre mapas interactivos donde se geolocaliza cada punto de interés o ubicación gestionada. El manual que recoge las pantallas de todas las funcionalidades puede encontrarse en el anexo “Manual de usuario”

Algunas de estas operaciones pueden ser realizadas en modo “sin conexión”, permitiendo la utilización de la aplicación en circunstancias en las que no existe

conectividad de datos para posteriormente sincronizar la información nueva o modificada por el usuario contra los servidores SGC[1] de MAPFRE cuando se reestablezca.

Éste proyecto fue llevado a cabo por Okode en agosto de 2014 para las tres plataformas móviles más populares del mercado (iOS, Android y Windows Phone) y se sigue usando en la actualidad.

Este Trabajo de Fin de Grado tiene como objetivo principal dar un enfoque y visión global acerca de su migración, que consiste en desarrollar una aplicación que recoja las funcionalidades del proyecto existente (**iMediador v.1.0** en adelante) y añada nuevas funcionalidades usando *Kony* como herramienta de desarrollo. *Kony* es una herramienta dirigida a entornos empresariales que facilita el desarrollo de aplicaciones multiplataforma. En el apartado 2.2 se explica en detalle.

Entre las nuevas funcionalidades encontramos la posibilidad de asignar gestiones a un mediador, la ordenación de las gestiones del plan de visitas, la geolocalización de la ubicación actual (para evitar introducir la dirección al crear un contacto, por ejemplo), la ampliación de los criterios de búsqueda de contactos y la capacidad de cerrar una gestión.

Puesto que la envergadura del proyecto es considerable, el equipo de trabajo estará formado por dos responsables de gestionar el proyecto y cuatro desarrolladores. Como miembro del equipo tendré que participar en la toma de decisiones, diseñar modelos conceptuales, desarrollar módulos de código y probar la aplicación.

Uno de los requisitos más importantes de **iMediador** (en cualquiera de sus versiones) es que la aplicación garantice la seguridad de que la información es inaccesible ante usuarios no autorizados mediante la utilización de diferentes mecanismos de autenticación, a la vez que ofrece al mediador una experiencia de usuario adecuada para no obligarle a identificarse completamente cada vez que necesita realizar cualquier acción a través de la aplicación.

1.2. Objetivos

1.2.1. Objetivo general

El objetivo principal es desarrollar una aplicación móvil con soporte multiplataforma para el colectivo de mediadores de MAPFRE que permita implementar una serie de funcionalidades típicas asociadas a su trabajo diario en la gestión de contactos, vencimientos y visitas comerciales, tomando como punto de partida **iMediador v1.0**. Adicionalmente, se ofrecerán nuevas funcionalidades y se mejorarán las existentes.

[1] Sistema de gestión comercial

La aplicación, tras ser desarrollada, será publicada de manera empresarial por MAPFRE para su descarga e instalación por parte del colectivo de mediadores.

1.2.2. Objetivos particulares

Los objetivos particulares perseguidos en este Trabajo de Fin de Grado son los siguientes:

- Análisis y modelado del dominio, con la incorporación de nuevos requisitos (asignación de gestiones y su ordenación en el plan de visitas, geolocalización de contactos, ampliación de los criterios de búsqueda de contactos, etc.).
- Diseño de la arquitectura, con especial hincapié en la interfaz de usuario, así como la navegación entre formularios.
- Implementación en un sólo lenguaje y posterior compilación a cada una de las plataformas soportadas (iOS, Android y Windows Phone).
- Configuración de los servicios necesarios para ser invocados a través de *MMobile* [1].
- Realización de pruebas y verificación de que la aplicación tiene un desempeño adecuado ante diferentes escenarios típicos.

1.3. Estructura del documento

El documento está organizado en siete capítulos:

En el capítulo 2, se realiza un pequeño análisis del estado desde el que partimos, **iMediador v1.0**.

En el capítulo 3, se presenta el ámbito de **iMediador v2.0**, el detalle de la infraestructura que compone su ecosistema y la visión general de sus requisitos tecnológicos.

En el capítulo 4, se plasman los requisitos funcionales y no funcionales demandados por el cliente.

En el capítulo 5, se describe la arquitectura usada en la aplicación acompañada de los aspectos relacionados con la seguridad.

En el capítulo 6, se detalla la implementación de la solución y se analiza el rendimiento de la aplicación final.

En el capítulo 7, se exponen las conclusiones finales y los trabajos futuros.

2. Estado del arte

2.1. iMediador como aplicación nativa

El producto **iMediador** en su versión 1.0 fue desarrollado en agosto de 2014 para Android, iOS y Windows Phone. Se optó por desarrollar una solución nativa para cada plataforma siendo su arquitectura, la que se muestra en la Ilustración 1.

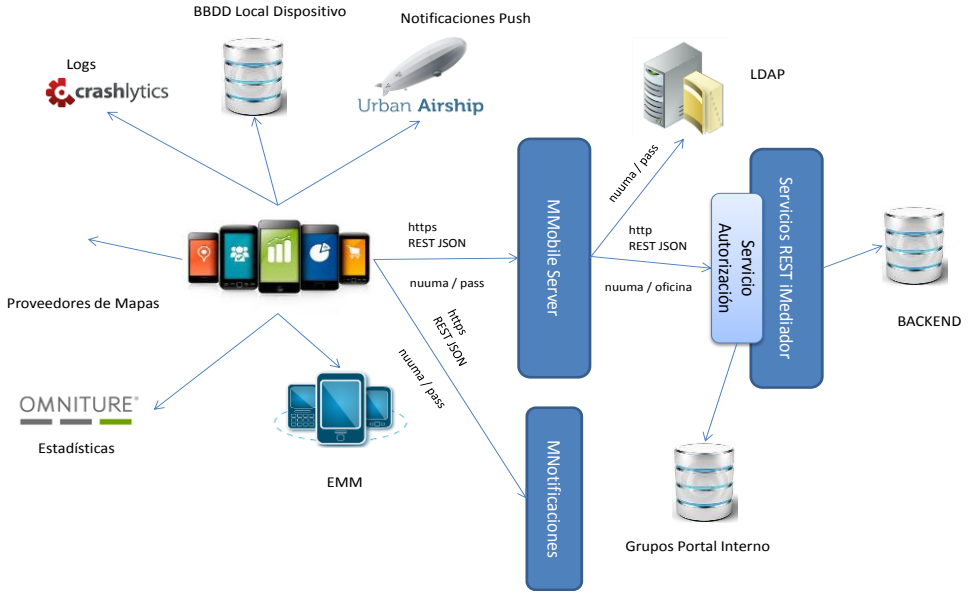


Ilustración 1: Arquitectura global de la app nativa (extraída de [8])

2.1.1. Requisitos de la aplicación nativa

Se implementó la lógica especificada en los documentos 'Requisitos Funcionales' y 'Requisitos No Funcionales' proporcionados por MAPFRE que se resumen a continuación [2] [3].

2.1.1.1. Requisitos funcionales

Login y niveles de acceso a la aplicación

La aplicación podrá ser utilizada con o sin conexión en función de la situación del dispositivo (cobertura), gestión MDM y decisión del usuario.

Menús y niveles de acceso

- Desconectado: sólo muestra las opciones para el uso desconectado de la aplicación.
- Sólo conectado: sólo muestra las opciones con acceso a información MAPFRE que implican conexión con servicios.
- Completa: muestra todas las opciones.

Acceso a la APP

- Para validar el que la aplicación esté siendo gestionada por el MDM, el API del mismo deberá ser capaz de determinarlo ya se encuentre el dispositivo con conexión o sin ella.

Elección de elemento de la estructura territorial

- Permitirá mostrar la lista de los elementos de la estructura territorial y seleccionar (sólo) uno. Para cada elemento se mostrará: Código, nombre y tipo de elemento (DT-DT-OD).

Caducidad de acceso

- La aplicación tendrá la capacidad para bloquearse tras un período de tiempo configurable de no utilización de la misma por parte del usuario.
- La configuración del intervalo de tiempo se realizará a través de la consola de *MMobile*.

Menú principal

Ajustes

- Consulta de perfil
- Modificación de PIN
- Notificaciones
- Aviso legal
- Versión

Perfil y configuración del usuario

- Perfil SGC
- PIN
- Notificaciones

Opciones de usuario

Gestiones

Punto del menú principal que da acceso a las gestiones:

- Bandeja de gestiones
    - Pantalla inicial
    - Pantalla lista de gestiones
- Mapa
    - Pantalla de detalle de una gestión
- Alta de gestión

Contactos

Punto del menú principal que da acceso a los contactos y a los vencimientos asociados a este:

- Búsqueda de contactos
    - Pantalla inicial
    - Pantalla lista de contactos
    - Pantalla de detalle de un contacto
- Vencimiento de un contacto
- Detalle de un vencimiento
- Alta de un vencimiento
- Modificación de contacto
- Alta de contacto

Plan de visitas

Punto del menú principal que permite acceder a la gestión de los planes de visita que el usuario hubiera almacenado con anterioridad:

- Lista de planes de visita
- Detalle del plan
- Mapa
- Detalle de gestión del plan de visitas

Sincronización

La sincronización permitirá enviar a los servidores MAPFRE (API de servicios) las actualizaciones recogidas sobre los planes de visitas tanto en las observaciones de las

gestiones como en las altas de nuevos vencimientos o en el alta de nuevas gestiones propias por parte del mediador.

Podrá solicitarse a nivel de gestión, plan o lista de planes.

Localización del vehículo

Permitirá implementar las siguientes funcionalidades:

- Establecer posición del vehículo.
- Aviso. La aplicación podrá enviar avisos al usuario siempre que se encuentre iniciada (aunque el usuario podrá tenerla en segundo plano). También será necesario que el usuario no deshabilite en su dispositivo la capacidad para que las aplicaciones actualicen su contenido en segundo plano. Los avisos se generarán mediante una notificación local al usuario o bien mediante la visualización de un *popup* en caso de encontrarse en primer plano.
- Localizar vehículo.

Integración con Plataforma de Notificaciones

La aplicación de **iMediador** se integrará con la Plataforma de Notificaciones de MAPFRE (se explica en el apartado 3.2).

Será posible enviar notificaciones personalizadas no enriquecidas a los usuarios a través del API Business de la Plataforma de Notificaciones.

Información local del Plan de Visitas

Se mantendrá en local en el dispositivo toda la información necesaria para la gestión de cada plan de visitas.

2.1.1.2. Requisitos no funcionales

Usabilidad – Experiencia de Usuario

Las aplicaciones móviles serán desarrolladas maximizando en lo posible la experiencia de usuario de manera que dispongan de una interfaz sencilla e intuitiva. De igual modo se promoverá que las interfaces sean acordes y homogéneas respecto al resto de aplicaciones de la plataforma nativa destino.

Tablets

Se soportarán exclusivamente *tablets* iOS (iPad) y Android. La interfaz de usuario de la aplicación será coherente entre su versión *tablet* y su versión *smartphone* (p.a. entre iPhone / iPad) de manera que se permita disponer de versiones específicas de las pantallas de *smartphone* en *tablets* atendiendo a los siguientes requisitos:

- MAPFRE deberá proveer la versión correspondiente a cada interfaz de usuario en cada dispositivo por lo que deberá proveer una versión específica para *tablet* y una específica para *smartphone* para cada pantalla que se desee adaptar.
- Dado que las aplicaciones que se desarrollarán son universales será necesario que la versión de la interfaz propuesta para cada dispositivo (*smartphone* / *tablet*) ofrezca el mismo funcional bajo los mismos controles visuales de manera que el controlador (lógica interna de la *app*) que gestiona la interfaz pueda ser el mismo para ambas versiones. Esto minimizará los costes futuros de mantenimiento y evolutivos de la aplicación.

Orientación vertical / horizontal de la interfaz de usuario

La orientación de las versiones *tablet* de la aplicación podrá ser vertical u horizontal. Para ello MAPFRE deberá proveer los diseños y guías de navegación de las diferentes orientaciones de la versión *tablet* de la aplicación (para cada plataforma soportada iOS / Android).

Comportamiento ante eventos externos

La aplicación gestionará correctamente su comportamiento ante eventos externos (llamada entrante, bloqueo por inactividad, etc.) volviendo al estado adecuado al recuperarse de dicha situación.

Aviso de no disponibilidad

La aplicación notificará adecuadamente al usuario (típicamente mediante alertas) ante situaciones de excepción como la imposibilidad de transmitir datos (p.a. por falta de cobertura) o fallo en la localización por acceso a los servicios de GPS.

Desconexión automática

La aplicación cerrará la sesión del usuario activo al pasar un tiempo de no interacción con el mismo. Este tiempo será configurable desde la consola de *MMobile*.

En caso de que la aplicación se encuentre ejecutando una tarea de sincronización con el servidor este periodo de tiempo no se tendrá en consideración, iniciándose por tanto al finalizar dicha tarea de sincronización.

Invocación a servicios

La aplicación controlará la respuesta de las invocaciones a los servicios informando al usuario con mensajes específicos de la forma adecuada.

Todas las invocaciones estarán parametrizadas por un *timeout* cuyo valor por defecto será configurable desde la consola de *MMobile* (un único valor compartido para todas las invocaciones que puedan realizarse desde la aplicación).

Permisos que se solicitarán al usuario

La aplicación solicitará al usuario los permisos necesarios para poder habilitar las siguientes funcionalidades:

- Recepción de notificaciones *PUSH*
- Acceso a la agenda de contactos
- Acceso a la galería de fotos
- Acceso a la información de localización (GPS)

Rendimiento

Las aplicaciones se desarrollarán vigilando siempre que el uso de la batería y la memoria sean los mínimos posibles.

Se generarán informes durante la fase de documentación acerca de los siguientes aspectos:

- Uso de la memoria
- Uso de la CPU
- Uso de Red

Función "volver" en la interfaz de usuario

Desde cualquier pantalla será posible volver a la pantalla anterior, bien mediante los botones habilitados al efecto en ciertas plataformas (Windows Phone, Android) bien mediante la inclusión de controles de navegación acordes con la plataforma actual (como en el caso de iOS donde típicamente existe un botón "volver" en cada uno de los controladores de navegación a la izquierda de su barra superior).

Integración con MMobile

Las aplicaciones se desarrollarán de manera integrada con *MMobile*, debiendo éstas ser registradas en la consola de la misma desde donde se podrá habilitar:

- Versionado de las aplicaciones
- Configuración personalizada
- Servicios mediados que se desee invocar
- Recepción de *logs*

Analíticas de Adobe Mobile Services

Las aplicaciones se integrarán con el API de *Adobe Mobile Services* [4] (se explica en el apartado 3.2) de manera que generen información analítica del uso de las mismas por parte de usuarios finales.

Será necesario que MAPFRE provea la guía de etiquetado de *Adobe Mobile Services* al inicio del Proyecto.

2.1.2. Arquitectura de la aplicación nativa

El desarrollo se basa en la pila nativa de cada plataforma y se divide en una arquitectura de capas respetando las buenas prácticas de separación de capas y siguiendo las buenas prácticas de *MMobile*:

- **Capa de presentación:** En esta capa residen las vistas y los controladores de estas. Se hace uso de los mecanismos para la definición de vistas de cada plataforma (*StoryBoards*, *Layouts XML*, *Layouts XAML*) así como de *MMobileForms*[2]. Las vistas se definen a partir de los temas y *layouts* base de cada plataforma facilitando al máximo posible cambios en el diseño en futuros evolutivos o adaptaciones de la aplicación. A través de la capa de negocio se recupera el modelo de datos a presentar o se envía para persistir las modificaciones de datos realizadas por el cliente final.

- **Capa de negocio:** En la capa de negocio se abstrae del resto de la aplicación la gestión de la lógica de negocio y su modelo asociado. El modelo de datos es tratado en este punto para tomar decisiones sobre el comportamiento de la app frente al usuario y se trasladan tanto estas decisiones (cambios de pantalla, registro de alertas, etc.) como el modelo de datos a la capa de presentación o a la capa de datos según si es necesario mostrar o persistir.

- **Capa de datos:** La capa de datos es la encargada de gestionar el acceso a datos, tanto para la persistencia en bases de datos locales como para el envío y recuperación desde el *backend* a través de los servicios vía HTTP. En el caso de las bases de datos locales, se usa *SQLite* [3] en las tres plataformas para homogeneizar las tecnologías utilizadas.

La solución está basada en el patrón modelo-vista-modelo de la vista (MVVM) [5] en el caso de Windows Phone, y en el modelo-vista-controlador (MVC) [6] en el caso de Android e iOS, aunque se intentó adaptar el MVVM a estas plataformas envolviendo los modelos con aquello que necesita la vista para facilitar la separación de los datos de la lógica de la aplicación y de la interfaz de usuario. En la Ilustración 2 podemos ver la aplicación del patrón MVC y la integración de *MMobileForms* en la aplicación.

[2] *MMobileForms* permite crear formularios de forma dinámica a partir de una definición de formulario en formato JSON [1].
[3] SQLite es un sistema de gestión de bases de datos relacional compatible con ACID [23].

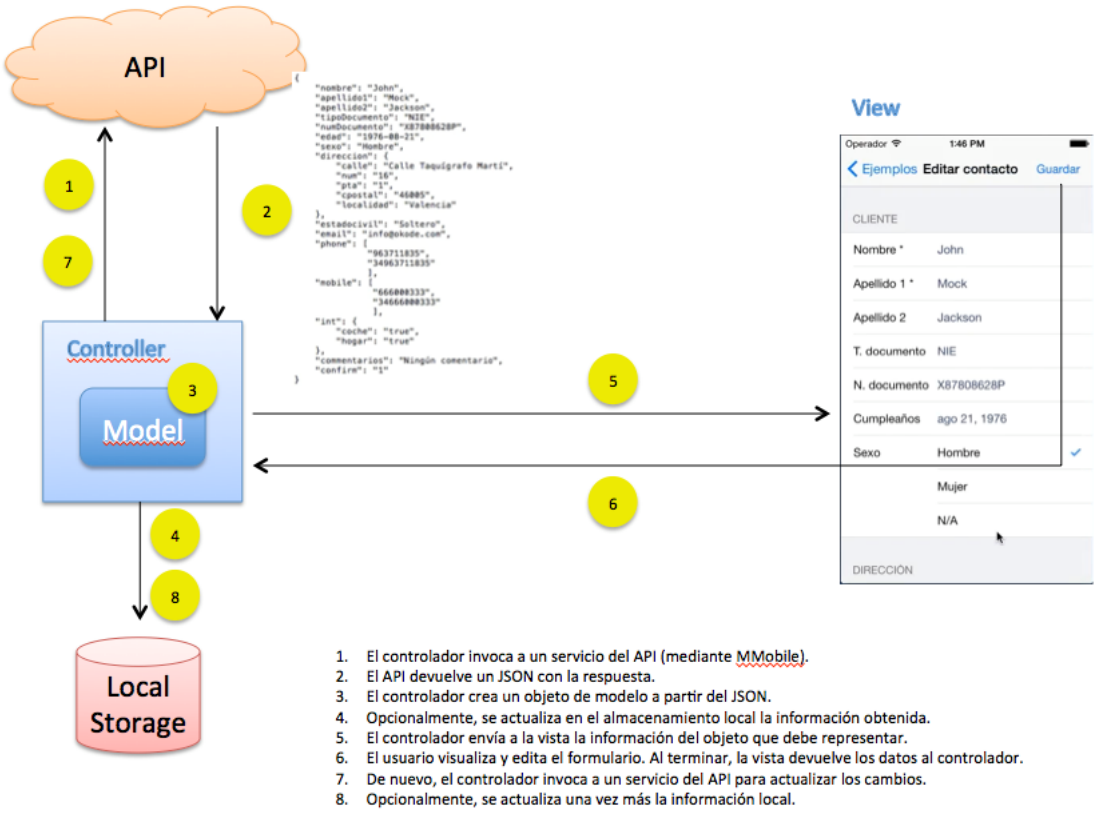


Ilustración 2: Diagrama de la arquitectura de la app nativa (extraída de [7])

En la Ilustración 3, se muestran las distintas capas en las que se estructura la aplicación y las relaciones entre ellas:

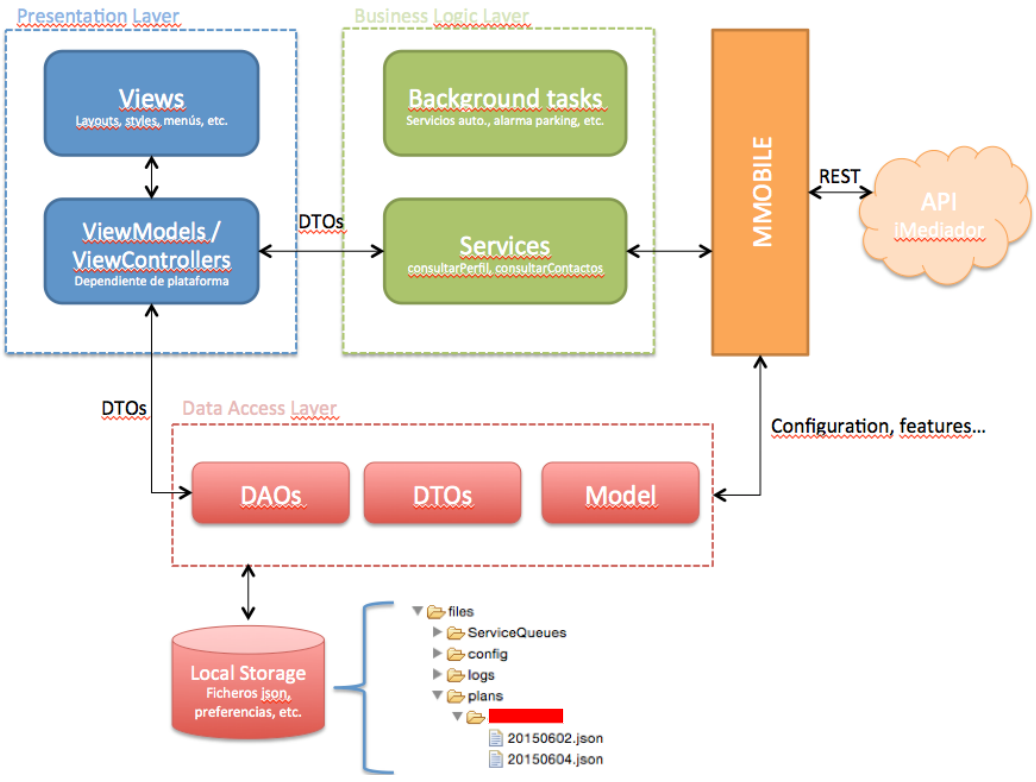


Ilustración 3: Capas de la app nativa (extraída de [7])

2.1.3. Particularidades de cada plataforma

A continuación, se comentan los aspectos específicos del desarrollo de la solución en cada plataforma.

2.1.3.1. Desarrollo de la solución en iOS

La solución para iOS se desarrolló en *Objective-C* y *Swift* estableciendo como target mínimo a iOS7 soportando la versión máxima disponible del operativo en el momento en el que se preparó el *bundle* para su distribución en los *markets*. Se soportan todos los dispositivos iOS que cumplan el requisito del sistema operativo.

La aplicación fue desarrollada como una aplicación universal, no se implementó una aplicación distinta para *smartphone* y *tablet*. Dado que iOS8 iba a estar disponible en los meses próximos al desarrollo y era necesario desarrollar con Xcode6, este fue el IDE/plataforma de desarrollo escogido.

2.1.3.2. Desarrollo de la solución en Android

La solución para Android se desarrolló en Java. No fue necesario desarrollar a más bajo nivel ninguno de los módulos dados los requisitos de la aplicación.

El API mínimo soportado fue el API 10 (Gingerbread) estableciendo como target el último estable disponible tal y como se recomienda en las guías de desarrollo de Android. Al igual que en iOS, se desarrolló una única aplicación para los distintos dispositivos Android. Se siguieron las guías de desarrollo para múltiples pantallas de la plataforma para dar soporte tanto a *smartphones* como a *tablets* a través de la configuración de los *layouts*.

2.1.3.3. Desarrollo de la solución en Windows Phone

La solución para Windows Phone se desarrolló en C# usando el *framework* de .NET para Windows Phone 8.1. En este caso, y a diferencia de las otras plataformas, se desarrolló una aplicación móvil con soporte exclusivamente para *smartphone* dado que no existían *tablets* Windows Phone 8.1

2.2. Plataforma de Movilidad *Kony*

Tal como publicitan en su web "*Kony* es la compañía de soluciones de movilidad empresarial basada en la nube que más rápido crecimiento ha tenido y un líder en el sector entre los proveedores de plataformas de desarrollo de aplicaciones móviles" [8].

Entre sus componentes destacan *Kony Visualizer* y *Kony Mobile Fabric*. El primero es un software WYSIWYM[4] para el diseño de aplicaciones nativas, web o híbridas con una base de código común que pueden ser ejecutadas en *smartphones*, *tablets*, escritorio y

[4] *What you see is what you mobilize*

wearables. El segundo actúa como solución *middleware* ofreciendo una infraestructura unificada de servicios basada en estándares que integra y soporta sistemas *backend*.

Por cuarto año consecutivo, se sigue considerando uno de los líderes en el Cuadrante Mágico de Gartner[5] en el ámbito de las *MADP*[6]. Gartner es una consultoría que se dedica a investigar la industria de las tecnologías de la información. Anualmente elabora una clasificación de fabricantes de tecnologías que denomina "Cuadrante Mágico" basándose en el estudio de las tendencias del mercado.

Debido a que facilita el diseño, compilación, despliegue y gestión de aplicaciones multiplataforma y al rápido crecimiento de su reputación, MAPFRE decide usar la Plataforma de Movilidad *Kony* como plataforma integrada para el desarrollo de aplicaciones móviles.

[5] Representación del resultado obtenido tras realizar el estudio del mercado.
[6] *Mobile Application Development Platforms*

3. iMediador v2.0

3.1. Ámbito

El objetivo principal es desarrollar una v2.0 de una aplicación existente que recoja las funcionalidades de la v1.0 y que incluya como nuevas la asignación de gestiones, la ordenación del plan de visitas, la geolocalización de direcciones, el cierre de gestiones y la ampliación de criterios en la búsqueda en contactos. Se define a continuación el marco de actuación del proyecto, tanto en función del negocio afectado como por su ámbito organizativo y geográfico.

3.1.1. Procesos de negocio afectados

El proyecto que se describe en este documento aportará mejoras en los procesos de:

- Asesoramiento y venta de seguros.

3.1.2. Entidades y unidades involucradas en el proyecto

Las entidades y áreas implicadas en el proyecto son:

- MAPFRE FAMILIAR
- DCTP / CCMI
- DCTP / CID

3.1.3. Usuarios a los que va dirigida la aplicación

Los usuarios de la aplicación objeto de este proyecto serán el colectivo de mediadores de MAPFRE.

3.1.4. Ámbito geográfico

El proyecto afecta a MAPFRE España y se prevé que la publicación de la aplicación se lleve a cabo de manera empresarial por parte de MAPFRE para su propio colectivo de mediadores. Es decir, no se llevará a cabo una publicación en los *markets* públicos de aplicaciones móviles.

3.2. Infraestructura

- ***Kony* Server**. Entorno mediador de los servicios externos proporcionados por el *backend*. La aplicación implementada en *Kony* obtiene la información necesaria del *backend* mediante la mediación de este servidor expuesto en internet.
- **Backend**. Los servicios externos a la arquitectura implementan la lógica de negocio de servidor a los que una aplicación móvil desarrollada bajo la arquitectura de *Kony* puede acceder. Este componente solo es accesible dentro de la red privada de MAPFRE. Estos servicios externos están implementados en base al protocolo REST, a su vez utiliza los medios de comunicación seguros o no (HTTPS / HTTP) y necesita autenticado de aplicación. Los servicios son mediados mediante el servidor *middleware* de *Kony*.
- **MMobile**: MBaaS[7] (*Mobile Backend as a Service*) que actúa como proxy ante las invocaciones a servicios ofreciendo configuración remota, configuración personalizada, activación de características, capacidad de bloqueo, reporte de *logs* y buenas prácticas.
    - Configuración remota: permite modificar la configuración de la aplicación sin necesidad de volver a compilar y distribuir la misma.
    - Configuración personalizada: dispone de un paquete para enviar datos arbitrarios a la aplicación.
    - Activación de características: posibilita la activación/desactivación de características.
    - Capacidad de bloqueo: permite deshabilitar versiones específicas de la app e incluso dispositivos concretos.
    - Reporte de *logs*: habilita el envío de *logs* para un usuario o un conjunto de usuarios ampliando la información ofrecida por los servicios de *crash reporting*.
    - Buenas prácticas: ofrece un código base no intrusivo para ayudar a los desarrolladores a seguir buenas prácticas de desarrollo.

[7] Modelo que permite vincular aplicaciones al almacenamiento en la nube, servicios analíticos y otras características como la gestión de usuarios, notificaciones *push* y la integración con redes sociales [22].

Todos los servicios ofrecidos por *MMobile* se encuentran implementados en el componente *MMobile,* que ofrece la posibilidad de realizar tareas comunes de alto valor añadido.

- **MPush** [1]: infraestructura software que facilita el envío de notificaciones *push* personalizadas. Desde la misma se pueden enviar notificaciones a un dispositivo concreto o a un conjunto de estos.

  Toda la gestión de notificaciones *push* se encuentra implementada en el componente *MPushListener* que ofrece la posibilidad de gestionar las notificaciones personalizadas que la aplicación pueda recibir.
- **Crashlytics** [9]: solución que registra las excepciones que se producen en tiempo de ejecución y genera un informe detallado con la traza de las mismas, además del modelo del dispositivo y la versión del sistema operativo. Resulta muy útil para corregir errores que no se han detectado en la fase de pruebas.
- **Adobe Mobile Services**: librería que permite capturar la actividad nativa de la aplicación (uso, comportamiento, gestos, etc.) y reenviar los datos a los servidores de Adobe para la realización de analíticas sobre el uso de la misma, así como la medición y optimización de la esta.
- **Jenkins** [10]: software de automatización extensible que se usa como servidor de integración continua[8]. Se basa en el uso de tuberías para encadenar tareas que, típicamente, suelen ser 'compilar, probar y entregar'. Puede usarse como eje para aplicar la entrega continua[9].
- **SonarQube** [11]: plataforma abierta para evaluar la calidad del código usando herramientas de análisis estático[10]. Como tal, cubre *los siete ejes de la calidad del código fuente* [12]. Además, permite la integración con Jenkins de tal manera que se realiza un análisis cada vez que este termina una compilación.
- **QUnit** [13]: *framework* para realizar pruebas unitarias[11] sobre código JavaScript.
- **Confluence** [14]: software de colaboración que se usa para intercambiar conocimiento entre los miembros de un equipo. Típicamente se usa a modo de wiki donde se agrupan manuales y documentos de interés facilitando la labor del desarrollo.
- **JIRA** [15]**:** software de gestión de incidencias que proporciona seguimiento de errores y problemas así como funciones de gestión de proyectos. Mediante la aplicación web se dan de alta las incidencias que reportan los usuarios responsables de realizar pruebas en la app ofreciendo un seguimiento de su estado.

[8] Práctica de desarrollo software donde los miembros de un equipo integran su trabajo frecuentemente.
[9] Capacidad para liberar software rápidamente en producción con la seguridad de que funcionará correctamente.
[10] Evaluación del código fuente sin necesidad de ponerlo en ejecución.
[11] Pruebas a las que se somete de un módulo de código para garantizar su correcto funcionamiento.

- **GitHub** [16]: plataforma de desarrollo colaborativo de software que hace uso del sistema de control de versiones git para almacenar proyectos. Además, proporciona control de acceso, seguimiento de errores, petición de funcionalidades, gestión de tareas y wikis.
- **Plastic** [17]: sistema de control de versiones[12] distribuido que simplifica las tareas de creación e integración de ramas. Además, trata de dar soporte al desarrollo paralelo, seguridad y desarrollo distribuido.

3.3. Entorno tecnológico

La solución desarrollada cumplirá los siguientes requisitos tecnológicos:

- Integración de sistemas: la aplicación se ajustará a la arquitectura actual de MAPFRE para el desarrollo de aplicaciones móviles. Concretamente se utilizarán los servicios de *MMobile*, *MPush*, *Crashlytics* (excepto en Windows Phone, ya que no está soportado) y *Adobe Mobile Services*.

- Escalabilidad del sistema: se facilitará la cobertura de las necesidades futuras de MAPFRE, tanto respecto a la incorporación de nuevos componentes reutilizables como a su evolución conforme a los cambios que se produzcan en las áreas de negocio de MAPFRE a las que la plataforma de movilidad dará soporte.

- Desacoplamiento entre capas: la solución presentará una clara separación entre los diferentes dominios o capas en que se estructura permitiendo de forma natural la evolución del sistema, así como la reutilización de componentes.

- Monitorización: se permitirá su monitorización gracias a un sistema de *logging* implementado a través del marco de referencia de movilidad *MMobile*. También podrán ser monitorizados los fallos en tiempo de ejecución (producción) a través del uso de *Crashlytics*.

- Modularización: la lógica de las aplicaciones se desarrollará aplicando criterios de modularidad a fin de favorecer la reutilización de código y mantenibilidad de las mismas.

- Rendimiento: el diseño de la solución se realizará valorando la optimización del rendimiento como un factor clave de éxito.

- Auditabilidad y trazabilidad lógica: la operativa de negocio realizada sobre la solución propuesta permitirá generar una traza auditable en soporte persistente que será recogida en la consola de administración de *MMobile*.

[12] Sistema que registra los cambios realizados sobre un fichero o un conjunto de ficheros de tal manera que se puede recuperar una versión específica.

- Multiidioma: las aplicaciones móviles tendrán su interfaz de usuario y mensajes en idioma español, aunque todas las cadenas de las aplicaciones estarán disponibles en archivos de recursos independientes que podrán ser traducidos a otros idiomas de manera sencilla cuando se requiera.

- Disponibilidad 24x365: el software desarrollado permitirá la operativa de negocio en condiciones de 24x365.

- Seguridad: el diseño de la solución propuesta garantizará la seguridad de la información almacenada en los sistemas de MAPFRE, de acuerdo al marco regulatorio vigente y normativa interna existente. Concretamente se utilizarán mecanismos de autenticación de usuarios finales que, en conjunción con el MDM de MAPFRE, permitirán habilitar la seguridad de las aplicaciones de manera adecuada.

- Encapsulamiento de datos: Las aplicaciones que se desarrollarán serán las responsables de administrar la integridad de sus propios datos. Concretamente toda la información sobre contactos, gestiones, planes de visitas e información de perfil estará accesible de manera exclusiva para la aplicación móvil que las gestiona.

4. Modelado Conceptual

4.1. Modelo de Casos de Uso

Al tratarse de una migración, se asumen los mismos requisitos funcionales y no funcionales que en la versión 1.0 (detallados en la sección 2.1.1) quedando el modo horizontal en *tablets* previsto para un evolutivo posterior. Además de los citados requisitos, se han añadido las funcionalidades que muestran en los diagramas de casos de uso (ilustraciones 4 a 8), tomando como notación el estándar UML [18].

Como miembro del equipo mi aportación ha sido la realización del modelo de casos de uso, el diagrama de clases y el modelo de navegación.

Gestiones sin asignar

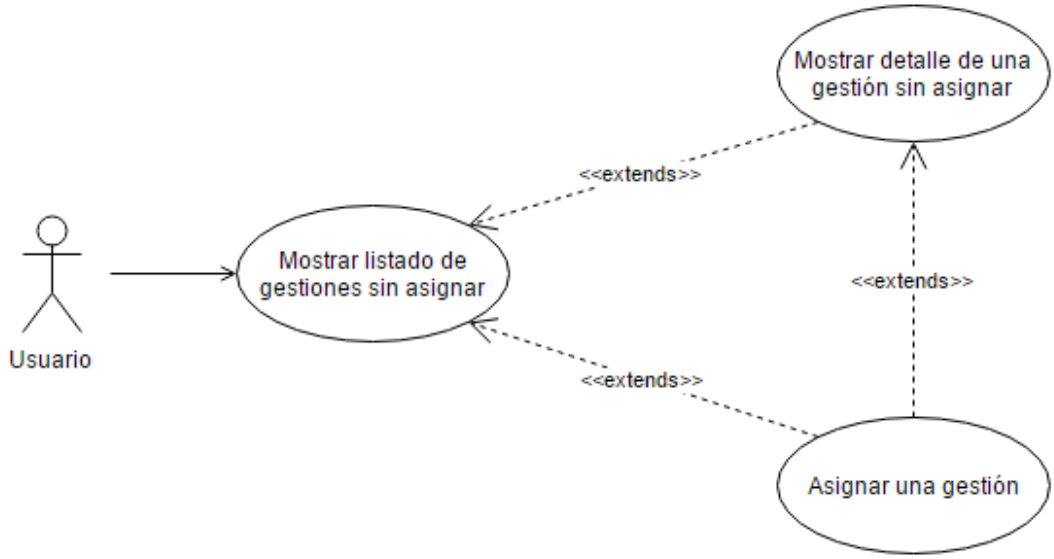


Ilustración 4: Gestiones sin asignar

Ordenación del plan de visitas

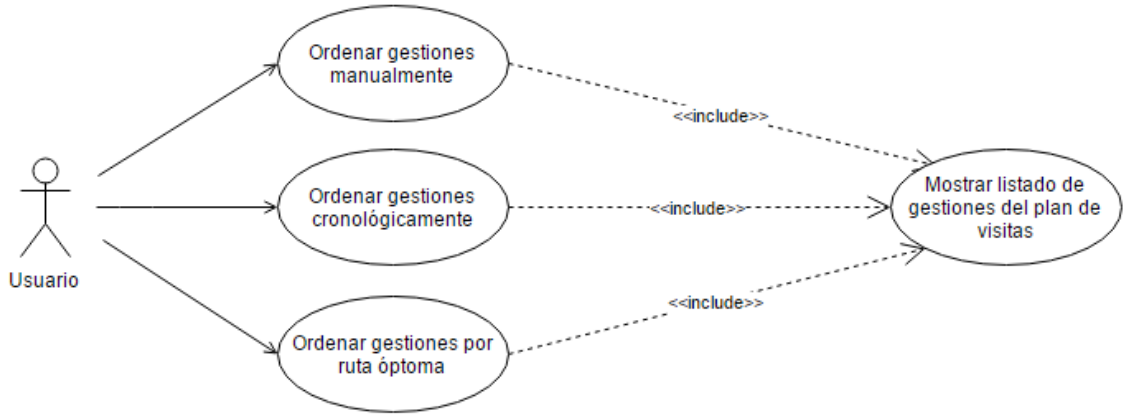


Ilustración 5: Ordenación del plan de visitas

Geolocalización de direcciones

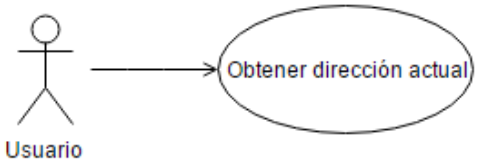


Ilustración 6: Geolocalización de direcciones

Cierre de gestiones

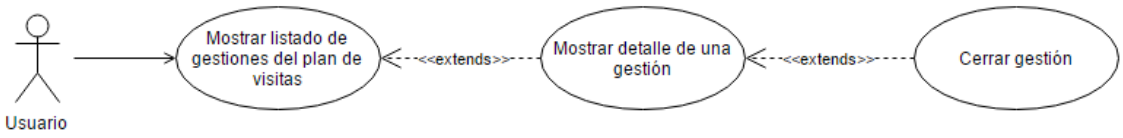


Ilustración 7: Cierre de gestiones

Ampliar criterios de búsqueda de contactos

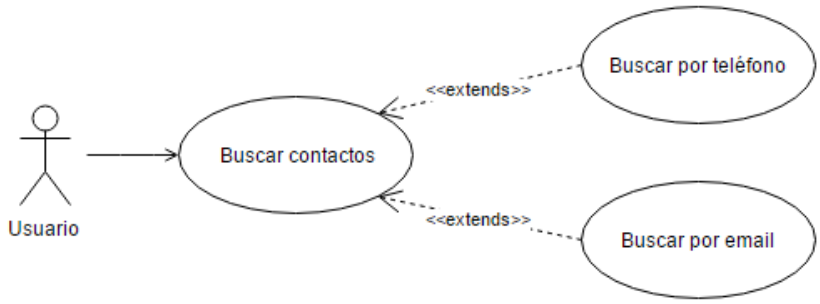


Ilustración 8: Ampliar criterios de búsqueda de contactos

Cada uno de los casos de uso mostrados en el diagrama se describen en las plantillas 1 a 10.

Identificador	CU01
Nombre	Mostrar listado de gestiones sin asignar.
Descripción	Muestra un listado de las gestiones sin asignar disponibles para el usuario actual.
Entrada	Identificador de usuario.
Proceso	Llamar al servicio que devuelve las gestiones sin asignar.
Salida	Listado de gestiones sin asignar.

Tabla 1: CU01 – Mostrar listado de gestiones sin asignar

Identificador	CU02
Nombre	Mostrar detalle de una gestión sin asignar.
Descripción	Muestra el detalle de una gestión sin asignar: datos del cliente, dirección, acción comercial y fecha de caducidad.
Entrada	Identificador de gestión.
Proceso	Llamar al servicio que devuelve el detalle de una gestión.
Salida	Detalle de la gestión.

Tabla 2: CU02 – Mostrar detalle de una gestión sin asignar

Identificador	CU03
Nombre	Asignar una gestión.
Descripción	Permite asignar una gestión que no esté asignada a otro usuario desde el listado de gestiones sin asignar o el detalle de una.
Entrada	Identificador de gestión, identificador de usuario al que asignar la gestión.
Proceso	Llamar al servicio que asigna una gestión a un usuario. Si la respuesta es correcta, vuelve a la pantalla de lista de gestiones actualizada. En caso contrario, informa del mismo y se queda en la pantalla por si el usuario quiere corregirlo.
Salida	Respuesta indicando que la operación se ha realizado correctamente o error en caso contrario.

Tabla 3: CU03 - Asignar una gestión

Identificador	CU04
Nombre	Ordenar gestiones manualmente.
Descripción	Se dará la posibilidad al usuario de ordenar las gestiones que están en el plan de visitas según el criterio que este considere.
Entrada	Listado de gestiones en el plan de visitas.
Proceso	Ordenar el listado de gestiones según desee el usuario.
Salida	Listado de gestiones en el plan de visitas ordenado.

Tabla 4: CU04 - Ordenar gestiones manualmente

Identificador	CU05
Nombre	Ordenar gestiones cronológicamente.
Descripción	Se dará la posibilidad al usuario de ordenar las gestiones que están en el plan de visitas según la hora que tengan asignada.
Entrada	Listado de gestiones en el plan de visitas.
Proceso	Ordenar el listado de gestiones de manera cronológica.
Salida	Listado de gestiones en el plan de visitas ordenado.

Tabla 5: CU05 - Ordenar gestiones cronológicamente

Identificador	CU06
Nombre	Ordenar gestiones por ruta óptima.
Descripción	Se dará la posibilidad al usuario de ordenar las gestiones de están en el plan de visitas según la mejor ruta posible.
Entrada	Listado de gestiones en el plan de visitas.
Proceso	Llamar al servicio de Google para calcular la mejor ruta. Ordenar el listado de gestiones en función de la respuesta recibida.
Salida	Listado de gestiones en el plan de visitas ordenado.

Tabla 6: CU06 - Ordenar gestiones por ruta óptima

Identificador	CU07
Nombre	Obtener dirección actual.
Descripción	Se obtendrá la dirección actual del dispositivo para facilitar al usuario su introducción en las pantalla de alta y modificación de un contacto.
Entrada	Ninguna.
Proceso	Llamar a la API para obtener las coordenadas actuales. Llamar al servicio de geolocalización inversa de Google.
Salida	Dirección actual.

Tabla 7: CU07 - Obtener dirección actual

Identificador	CU08
Nombre	Cerrar gestión.
Descripción	Se permitirá al usuario cerrar las gestiones.
Entrada	Identificador de la gestión.
Proceso	Llamar al servicio de cierre de gestiones.
Salida	Respuesta del servicio indicando si se ha cerrado la gestión o ha ocurrido algún error.

Tabla 8: CU08 - Cerrar gestión

Identificador	CU09
Nombre	Buscar por teléfono.
Descripción	Se ofrecerá la posibilidad de buscar contactos por teléfono. Este criterio es opcional.
Entrada	Número de teléfono.
Proceso	Llamada al servicio de consulta de contactos.
Salida	Listado de contactos cuyo teléfono sea el indicado.

Tabla 9: CU09 – Buscar por teléfono

Identificador	CU10
Nombre	Buscar por email.
Descripción	Se ofrecerá la posibilidad de buscar contactos por email. Este criterio es opcional.
Entrada	Email.
Proceso	Llamada al servicio de consulta de contactos.
Salida	Listado de contactos cuyo email sea el indicado.

Tabla 10: CU10 - Buscar por email

4.2. Diagrama de Clases

El modelo de clases que intenta recoger la información necesaria proporcionada por los servicios y adaptarla a lo que esperan recibir, es el que se muestra en la Ilustración 9.

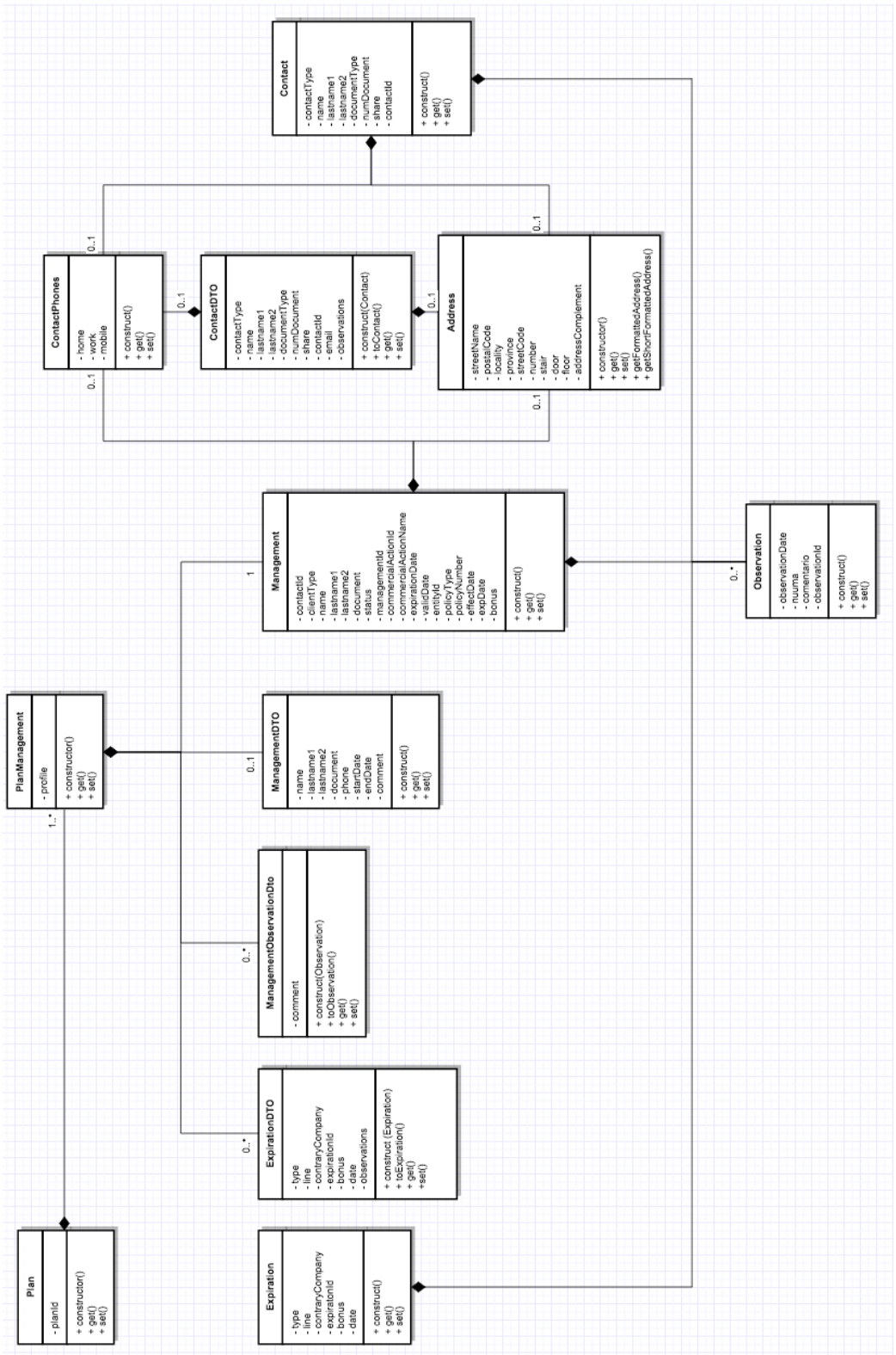


Ilustración 9: Diagrama de clases de la aplicación

4.3. Modelo de Navegación

La navegación de la aplicación se resume en las ilustraciones 10 a 15.

Ilustración 10: Modelo de navegación - Menú principal

Gestiones (1) representa las gestiones asignadas a un mediador y Gestiones (2) las que están pendientes de asignar.

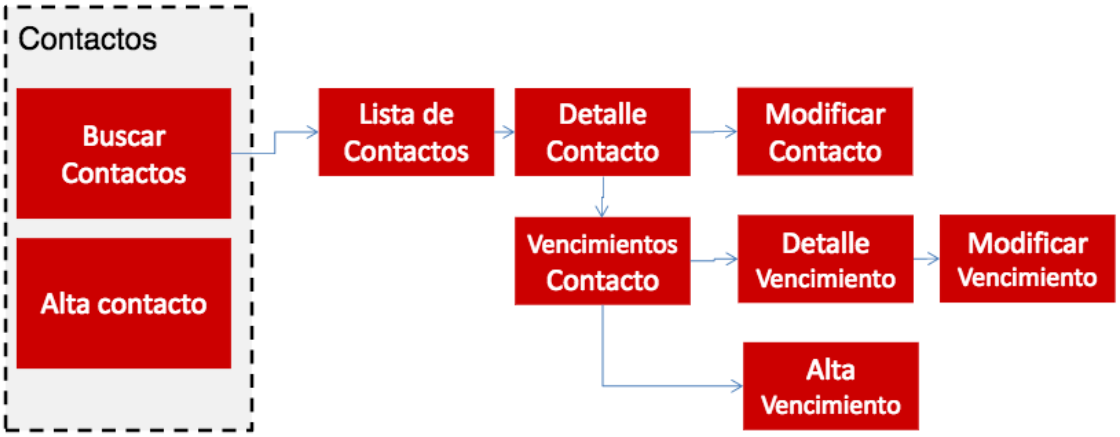


Ilustración 11: Modelo de navegación - Contactos

Ilustración 12: Modelo de navegación - Gestiones

El detalle de contacto (1) enlaza con la pantalla de detalle del contacto de la ilustración 11.

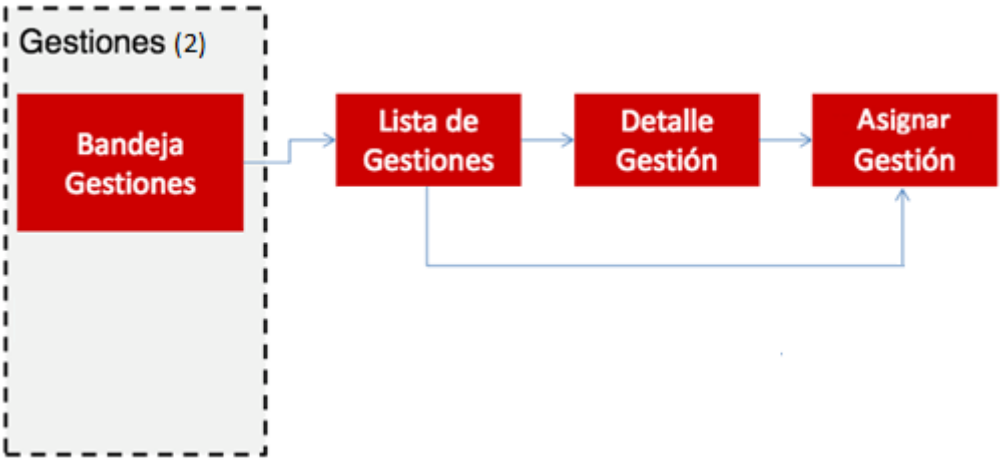


Ilustración 13: Modelo de navegación - Gestiones sin asignar

Ilustración 14: Modelo de navegación - Plan de visitas

Vencimientos del contacto (1) enlaza con la pantalla de vencimientos del contacto de la ilustración 11, pero con distinta funcionalidad según el modo de conexión.

Ilustración 15: Modelo de navegación - Parking

5. Diseño del sistema

5.1. Arquitectura de la aplicación

5.1.1. Introducción

iMediador v2.0 sigue una arquitectura de tres capas [18].

- **Capa de presentación**. Las vistas están implementadas con *MMobileForms,* basado en html, js y css; y por formularios basados en componentes de *Kony*. Ambos tipos de vista son reutilizables entre las tres plataformas. Las vistas se gestionan a través de un controlador de navegación. Cada vista tiene un controlador y un *ViewModel* asociados y la comunicación entre la lógica de negocio del controlador y la vista se realiza a través del *ViewModel*. Los objetos de *ViewModel* son los únicos que modifican la vista, tanto para reflejar datos como para algunas acciones de maquetación extraordinarias/dinámicas.
- **Lógica de negocio**. Se trata del código que reside en los móviles y que se ejecuta en el ámbito de un dispositivo móvil en manos del usuario final. La aplicación sigue un patrón MVVM en el cual la lógica de la aplicación reside en un controlador que se encarga de unir capa de datos y presentación a través de un *ViewModel*.

- **Capa de datos.** La capa de datos se compone de una serie de modelos junto a la lógica de invocación de servicios. Los modelos de la aplicación se han codificado con *prototype* para facilitar una gestión más orientada a objetos y se usan tanto para encapsulamiento como para transformación de datos (DTOs [20] principalmente).

5.1.2. Diagrama de la arquitectura

En la Ilustración 16 se muestra el diagrama de arquitectura global con los distintos componentes implicados en la solución:

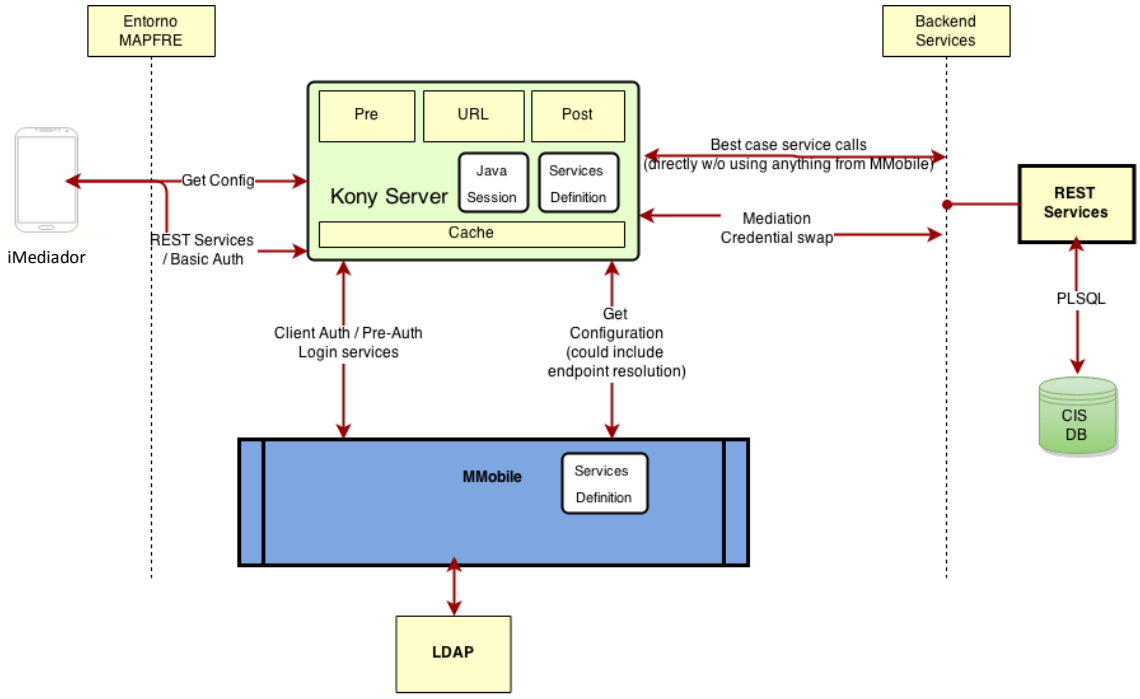


Ilustración 16: Diagrama de la arquitectura global (extraído de [1])

A nivel de app, se ha orientado hacia una modificación del patrón Modelo-Vista-Modelo de Vista (MVVM) en el que se recupera el uso de controladores similares a los del patrón MVC (Modelo-Vista-Controlador) pero en el que el *ViewModel* no solo mantiene el estado de la vista, sino que la lógica de interacción con la vista es semejante al patrón MVP (Modelo-Vista-Presentador) [20]. Estas modificaciones eran necesarias en proyectos *Kony* para cubrir la falta de herramientas en el *framework* en su versión 6.5 para la gestión de la navegación o los *binding* automáticos.

El objetivo principal era usar un patrón para la gestión de los formularios de *Kony* en el que se consiguiera una buena separación de conceptos. En la Ilustración 17 podemos ver la aplicación de dicho patrón en la app para gestionar el ciclo de un formulario.

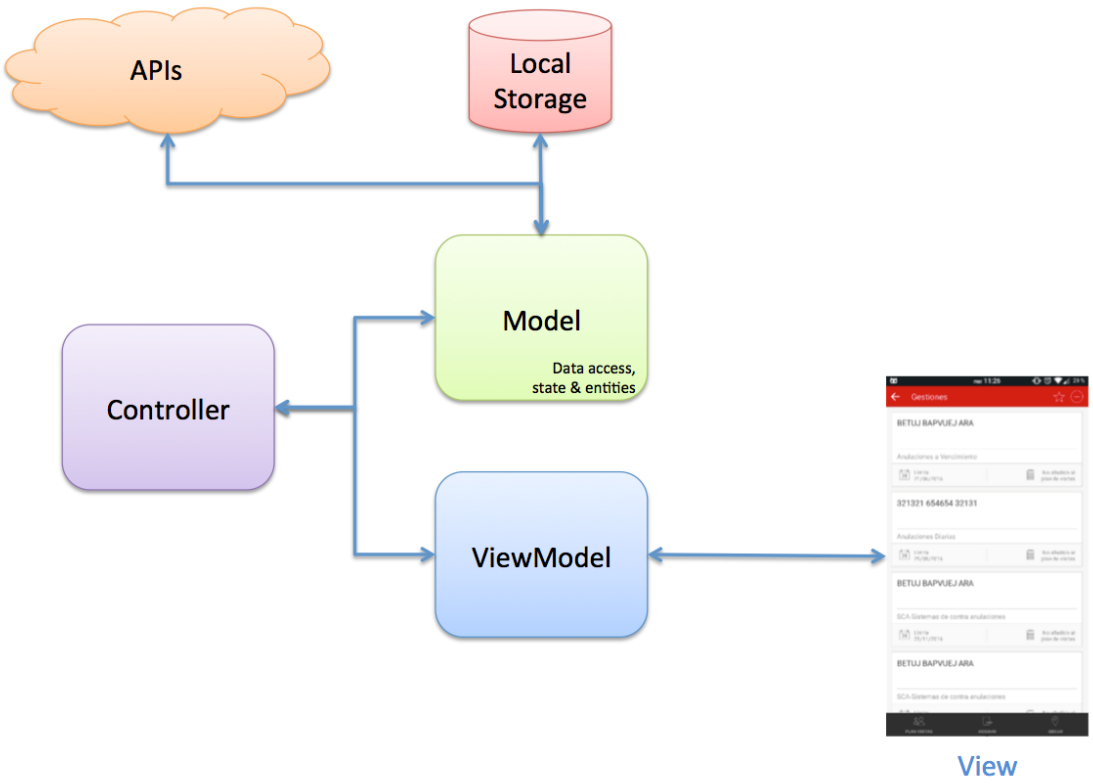


Ilustración 17: Diagrama de la arquitectura

Controller

Se encarga de gestionar la lógica de negocio, la navegación y hace de puente entre la capa de datos y la de presentación.

Es importante destacar que, para separar la lógica de negocio de la presentación y facilitar la abstracción de los elementos de la arquitectura de la aplicación, se ha cambiado el modelo de navegación de *Kony*. Aunque es evidente que la navegación termina invocando al método "*show*" del formulario (mecanismo de navegación de *Kony*).

En la aplicación se navega entre controladores. El controlador de navegación es el que se encarga de instanciar el controlador, buscar el formulario asociado e invocar a la navegación de *Kony* per se además de muchos otros ajustes y mejoras sobre el ciclo de vida.

Existe un controlador por formulario que en base a la lógica de negocio invoca a la capa de modelo para que recupere datos de servicios o local para trasladarla posteriormente al *ViewModel*.

Habitualmente se divide el código en inicialización, eventos de ciclo de vida del formulario, eventos de lógica de negocio de la IU, eventos de navegación y lógica interna como se puede observar en la Ilustración 18.


```
/**
 * frmClientSearchController
 * ..........
 */
function FrmClientSearchController() {…
};

//Extend from BaseController
FrmClientSearchController.prototype = Object.create(BaseController.prototype);
FrmClientSearchController.prototype.constructor = FrmClientSearchController;

/**
 * FORM LIFECYCLE
 * ----------
 *
 */
FrmClientSearchController.prototype.init = function () {…
};

/**
 * ACTION EVENTS
 * ----------
 */
FrmClientSearchController.prototype.onSearch = function(clientSearch) {…
};

/**
 * NAVIGATION
 * ----------
 */
FrmClientSearchController.prototype.navigateToClientList = function(clients) {…
};

FrmClientSearchController.prototype.navigateToClientDetail = function(client) {…
};

/**
 * CONTROLLER LOGIC
 * ----------
 */
FrmClientSearchController.prototype.invokeClientSearch = function (message, clientSearch) {…
};

FrmClientSearchController.prototype.clientSearchCallback = function (result) {…
};
```

Ilustración 18: Código del controlador de 'búsqueda de clientes'

Model

La capa de modelo está compuesta por las entidades, en las que se han definido las clases a través de *prototype*; la capa de acceso a datos (servicios y disco) y que mantiene el estado que el controlador propagará en caso de necesitar hacerlo en la navegación.

En los modelos también se han usado algunos DTOs necesarios dado que algunos APIs utilizados no son *model-driven* y definen modelos de entrada y salida distintos entre servicios y operaciones. Las entidades también se basan en *prototype* para definir el tipo de objeto. En la Ilustración 19 podemos ver un ejemplo de cómo se ha gestionado la entidad Cliente.

modelClient.js

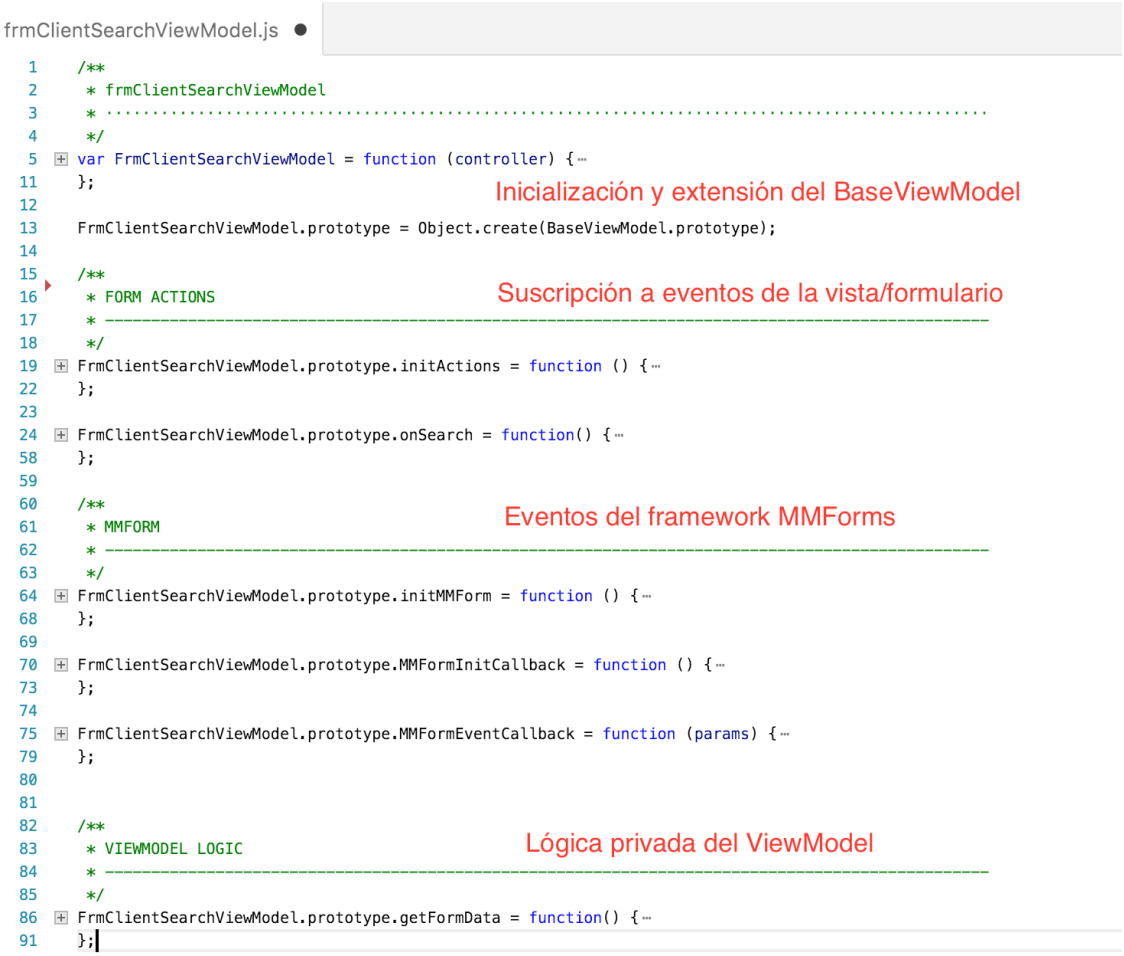
```
/*
 * Client is something similar to Contact
 * It is used on Management and it holds the data of the Contact which is associated to it
 * However, they are not the same concept. The client seems to be a copy of the contact when the
 * management was created
 */
var Client = function (client) {
    if (client) {…
    } else {…
    }
};

Client.prototype.populateFromService = function (serviceObject) {…
};

Client.prototype.populateFromForm = function (formObject) {…
};

Client.prototype.toServiceData = Client.prototype.toFormData = function () {…
};

Client.prototype.toServiceMapping = Client.prototype.toFormMapping = {…
};

Client.prototype.getFullName = function () {…
};

//InsuranceCalculator
Client.prototype.toInsuranceCalculator = function () {…
};
```

Ilustración 19: Modelo de la entidad Cliente

ViewModel

Los *ViewModel* mantienen el estado de la vista y los comandos ejecutables desde la interfaz. Para ello, se suscriben a los eventos de la vista y si se ejecuta un comando (p.e: seleccionar un elemento de una lista) se dispara un evento hacia el controlador para que este actúe en base a la lógica de negocio.

Al igual que en el caso de los controladores, existe una instancia de un *ViewModel* por vista/formulario. Cualquier lógica relacionada con la vista y su estado sólo debe aparecer en estas instancias. En la Ilustración 20 se muestra un ejemplo del *ViewModel* del formulario de 'búsqueda de clientes'.




```
/**
 * frmClientSearchViewModel
 * ·····
 */
var FrmClientSearchViewModel = function (controller) {…
};

FrmClientSearchViewModel.prototype = Object.create(BaseViewModel.prototype);

/**
 * FORM ACTIONS
 * ----------
 */
FrmClientSearchViewModel.prototype.initActions = function () {…
};

FrmClientSearchViewModel.prototype.onSearch = function() {…
};

/**
 * MMFORM
 * ----------
 */
FrmClientSearchViewModel.prototype.initMMForm = function () {…
};

FrmClientSearchViewModel.prototype.MMFormInitCallback = function () {…
};

FrmClientSearchViewModel.prototype.MMFormEventCallback = function (params) {…
};


/**
 * VIEWMODEL LOGIC
 * ----------
 */
FrmClientSearchViewModel.prototype.getFormData = function() {…
};
```

Ilustración 20: Código del viewModel de 'búsqueda de clientes'

Aunque fui partícipe en las conversaciones relacionadas con la arquitectura de la aplicación, su diseño no estaba entre mis responsabilidades, siendo la decisión final del resto de miembros del equipo (especialmente del Director Técnico) debido a que su experiencia era mucho mayor que la mía.

5.2. Capa de servicios

La capa de servicios está constituida por los servicios desarrollados por MAPFRE para sincronizar la información recogida en el dispositivo por parte de los mediadores con el entorno de servidor de MAPFRE.

A nivel de la aplicación móvil, a esta capa se accede a través del *Kony Middleware,* siendo responsabilidad de MAPFRE que dichos servicios sean alcanzables.

5.2.1. Servicio de mapas y geolocalización

Se utilizan los servicios de mapas y geolocalización del API de Google Maps tanto en Android como en iOS para homogeneizar los resultados entre las dos plataformas. En el

caso de Windows Phone es necesario utilizar su propia solución dado que no existe SDK de Google Maps ni, por licencia, se pueden utilizar datos recuperados de Google para mostrarlos en mapas que no sean provistos por este.

Las llamadas relacionadas con la ubicación (geolocalización directa e inversa) se median a través de MMobile.

5.2.2. Modelo de comunicación con sistemas externos

Toda comunicación con sistemas externos a la app se realiza de forma asíncrona, sin bloquear el hilo principal de ejecución para evitar que el sistema operativo penalice a la aplicación. Si bien es cierto que en escenarios como la obtención de listados o detalles se bloquea la interacción con el usuario (a nivel de capa de presentación) esto no significa que la llamada se realice de forma síncrona.

5.3. Seguridad

Se siguen las guías de seguridad provistas por MAPFRE [1]. A continuación, se enumeran los mecanismos de autenticado y autorización implicados en la aplicación.

Acceso a *backend*

Toda comunicación con el *backend* se realiza a través de protocolo seguro (HTTPS). Dependiendo del servicio (si es público o privado) se requiere que la invocación del servicio se haga bajo autenticación.

Los elementos del *backend* se basan en autenticado por NUUMA/*password* contra el LDAP corporativo de MAPFRE. Dado que el *middleware* de *Kony* soporta este tipo de autenticado, los servicios se mediarán para el acceso desde los dispositivos móviles.

Sistemas externos

Los sistemas externos con los que se integra la aplicación conllevan sus propios mecanismos de autenticado/autorización ya implementados en los SDK provistos.

Por ejemplo, para el uso de la licencia *business* de Google Maps es necesario crear una clave de API y asociarla a la aplicación vía paquete o *AppID* y una firma generada a partir de la clave/certificado de publicación.

PIN de desbloqueo

Los requisitos incluyen la posibilidad de que el usuario defina un PIN de desbloqueo de la app para que esta use las credenciales almacenadas. Tanto el PIN como las credenciales no se almacenan en plano en los recursos de la aplicación, se utilizan los mecanismos de seguridad provistos por la plataforma para tal efecto.

6. Desarrollo de la solución

6.1. Introducción

Se ha llevado a cabo el desarrollo de una única solución mediante la **Plataforma de Movilidad *Kony*** (impuesta directamente por MAPRE), que implementa todos los requisitos especificados en el apartado 4.1.

Durante el reparto de tareas se asignaron las funcionalidades de la versión 1 a dos miembros del equipo que habían participado en el desarrollo de la aplicación nativa. El resto nos fueron asignadas al otro miembro y a mí, siendo responsabilidad mía la implementación de los casos de uso 4, 7 y 8.

Como resultado del se han obtenido tres aplicaciones:

- **Aplicación iOS**
  - Se soportan dispositivos iPhone o iPad con iOS 7.0 o superior.

- **Aplicación Android**
  - Se soportan *smartphones* o *tablets* con Android 4.0.3 (API 14) o superior.

- **Aplicación Windows Phone**
  - Se soportan *smartphones* con Windows Phone 8.1 Silverlight.

6.2. Estructura del proyecto

En la Ilustración 21 se detalla cual es la estructura de directorios de un proyecto implementado con *Kony*.

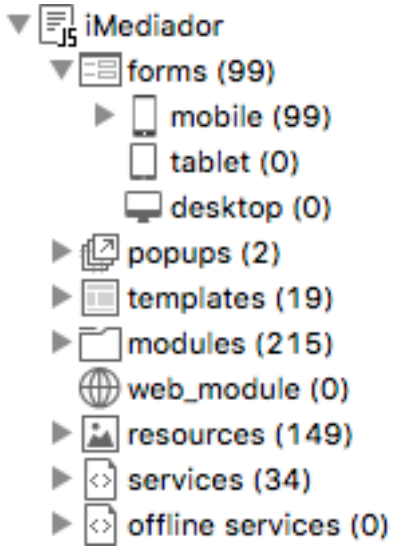


Ilustración 21: Estructura general del proyecto

- *forms*: formularios que representan las vistas de la aplicación generados mediante la aplicación *Kony Visualizer*.
- *popups*: ventanas modales.
- *templates*: plantillas de componentes visuales que pueden reutilizarse en toda la aplicación.
- *modules* → *js*: todos los ficheros JavaScript.
- *resources*: recursos compartidos (imágenes, iconos…).
- *services*: definición de servicios XML, SOAP, JAVA y JSON.

Como podemos observar en la Ilustración 22, en la sección *forms* → *mobile* están los formularios que representan las vistas de la aplicación.

Ilustración 22: Formularios de la app

En el caso de **iMediador v2.0**, dentro de *js* tenemos los modelos de la vista y sus controladores, los modelos de datos y toda la lógica de negocio de la aplicación. En las ilustraciones 23 y 24 se pueden ver todos los ficheros js que componen la aplicación.

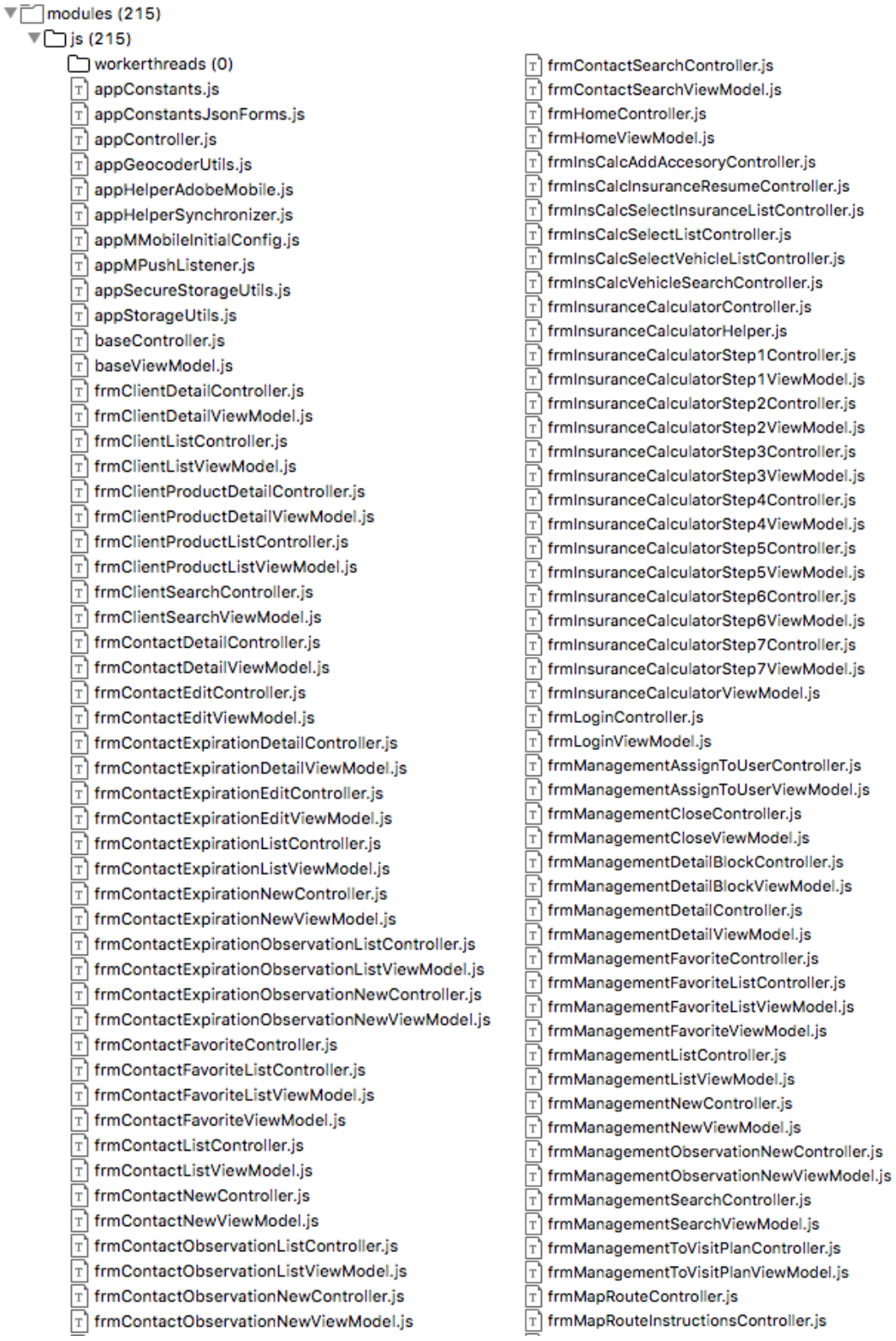


Ilustración 23: Ficheros de código fuente (1 de 2)

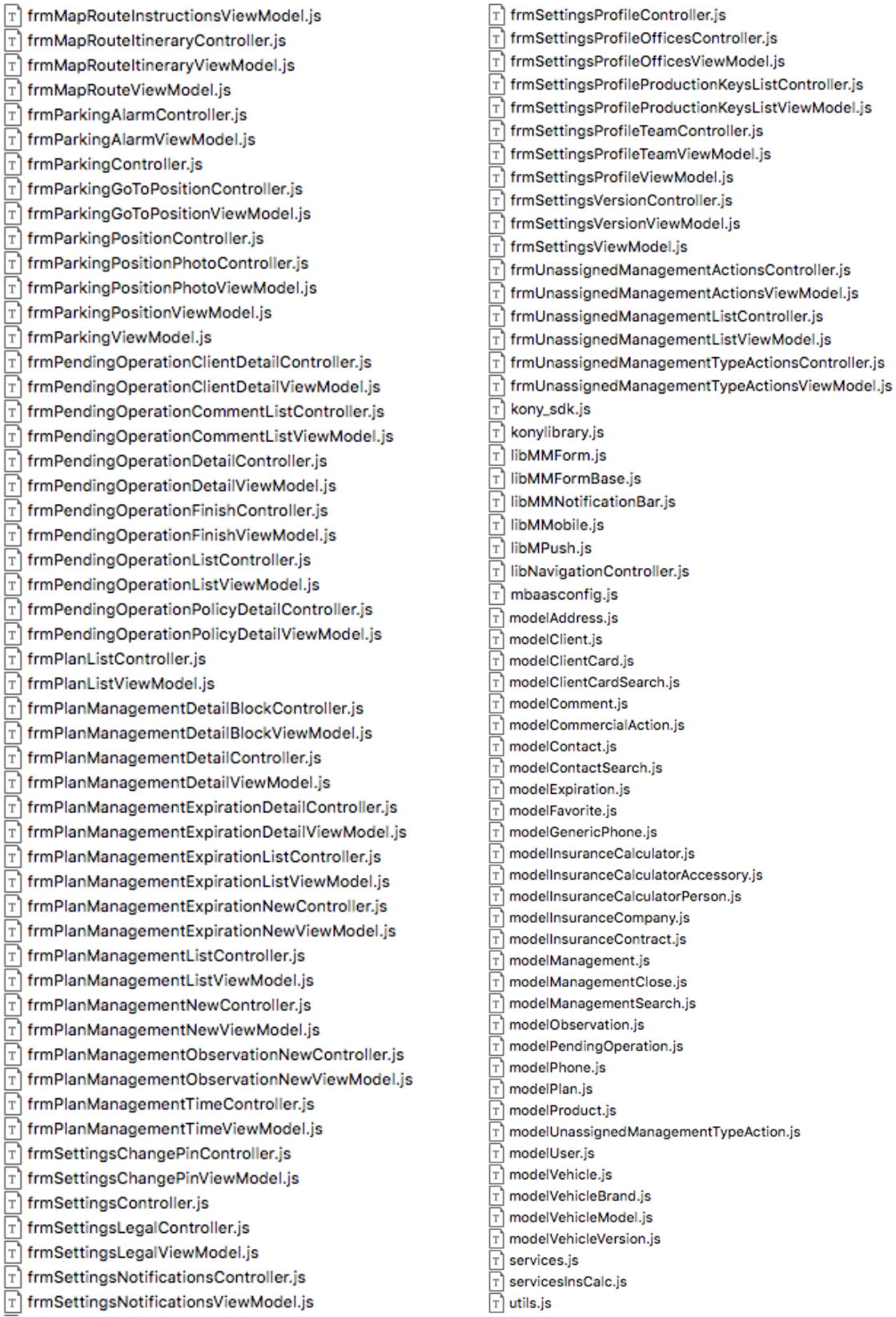


Ilustración 24: Ficheros de código fuente (2 de 2)

Puesto que no se pueden crear carpetas para organizar el proyecto, ésta es la convención de prefijos que se ha utilizado para tal efecto:

- app: componentes que se usan en toda la aplicación pero que agrupamos por su cometido. Por ejemplo, *appStorageUtils.js* contiene todas las utilidades relacionadas con persistencia en disco.
- base: controlador y modelo de vista genéricos de los que extenderán todos los controladores y modelos de vista usados en la aplicación.
- frm: cada formulario tiene su propio controlador y modelo de vista. Estos se encargan de la inicialización, las llamadas a servicios, el tratamiento de los resultados de las mismas, la gestión ante eventos, la lógica propia del formulario y la navegación entre ellos.
  Para organizarlo se ha optado por añadir a continuación de 'frm' la palabra clave que identifica el funcional. Así, los ficheros relacionados con las gestiones, por ejemplo, tienen el prefijo *frmManagement*.
- lib: librerías que se pueden reutilizar en otras aplicaciones (*MMForms*, *MPush*, *MMobile*, etc.)
- model: modelos de datos que encapsulan las propiedades y las transformaciones necesarias.
- services: invocación a servicios y mapeo sus resultados.
- utils: utilidades que aplican en múltiples partes de la aplicación, pero no se pueden agrupar en un fichero app*.js.

6.3. Dependencias

El proyecto de **iMediador v2.0** utiliza varias herramientas implementadas por terceros o implementadas en otros proyectos, como puede ser *MMobile*, *Mpush*, *Adobe Mobile* o *Crashlytics*. Estas implementaciones generalmente están incluidas mediante un FFI[13].

Dependencias sobre librerías externas

Sin considerar las dependencias principales incluidas por *Kony* y sin ánimo de listar todas las dependencias transitivas, **iMediador v2.0** tiene las siguientes dependencias con librerías externas:

- Kit de MPush
  - Android e iOS: mpush-kit-1.6
- *Reporting*
  - Android: answers-1.3.1
  - Android: beta-1.1.3
  - Android: crashlytics-2.5.1
  - Android: crashlytics-core-2.3.4
  - Android: fabric-1.3.5
  - iOS: Fabric-1.6.7
  - iOS: Crashlytics-3.7.1
- Analíticas
  - Android: adobemobile-4.5.4
  - iOS: adobemobile-ffi-ios (AdobeMobileSDK 4.6.1)

6.3.1. Notificaciones push

Tal y como ya se ha comentado en el apartado 3.2, la plataforma de notificaciones de MAPFRE (*MPush*) permite la gestión de notificaciones y la gestión del usuario en la plataforma.

En **iMediador v2.0** se hace uso de ellas para notificar a los usuarios de que se les han asignado nuevas gestiones.

Para la recepción de las mismas, es necesario iniciar sesión en la aplicación con un usuario y activarlas durante el proceso de *login* o bien acceder a la pantalla de ajustes > notificaciones. En la Ilustración 25 se muestran ambas opciones.

[13] *Foreign Functional Interface*. *Kony* permite extender su funcionalidad mediante la invocación a librerías nativas.

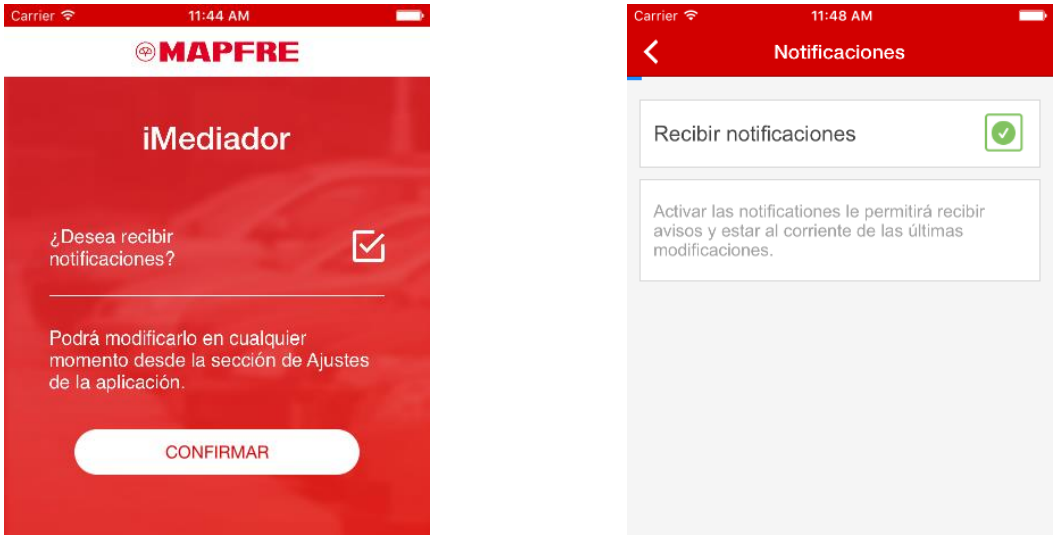


Ilustración 25: Activación de notificaciones

Para el envío de notificaciones personalizadas se debe hacer uso de la plataforma de notificaciones *MPush*, desde la cual podemos enviar notificaciones a un usuario en concreto.

Cuando la notificación se recibe en el dispositivo, se muestra un *popup* con la información recibida si la app está en ejecución. En caso de estar cerrada o en segundo plano, esta se abre previamente.

6.3.2. Estadisticas

Se usa el SDK de *Adobe Mobile Services* para el envío a los repositorios de desarrollo y producción de las estadísticas que se generen por usuario. Se ha seguido la guía de etiquetado [21] proporcionada por MAPFRE para la generación de eventos.

6.4. Gestión y configuración de la aplicación en remoto

Se han seguido las guías de *MMobile* para que **iMediador v2.0** pueda gestionar las versiones desplegadas en producción, añadiendo estas capacidades de configuración:

- Configuración de la caché
- Configuración de colas y *timeout* general
- Configuración personalizada (se detalla en la sección 6.4.1)
- Control de dispositivos bloqueados
- Control de activación/desactivación de la versión de la app en ejecución
- Control de activación/desactivación de funcionalidades (descritas en la sección 6.4.2)
- Envío de *logs*

6.4.1. *Valores configurables desde MMobile*

A continuación, se listan algunos de los valores configurables a través de *MMobile*:

- **sessionTimeout**: Tiempo en segundos tras el cual se debe cerrar la sesión de la aplicación si no se interactúa con ella.
- **disabledVersionText:** Texto a mostrar cuando la versión de la aplicación se desactiva.
- **disabledVersionLink:** Enlace opcional para facilitar a los usuarios que descarguen la nueva versión de la aplicación cuando se desactiva la actual.
- **serviceTimeoutSeconds:** Tiempo en segundos tras el cual se considera que se ha excedido el tiempo de espera.
- **managementStatus:** Posibles estados de una gestión.
- **expirations:** Tipos de vencimiento**.**
- **identificationTypes:** Tipos de documento.
- **phoneTypes:** Tipos de teléfono.
- **contactTypes.** Tipos de contacto.
- **provinces:** Conjunto de códigos de provincia.
- **roads:** Conjunto de los tipos de vía.

6.4.2. *Funcionalidades configurables desde MMobile*

A continuación, se listan los funcionales que pueden ser habilitados/deshabilitados desde *MMobile*:

- **managements:** opción *Gestiones* del menú principal.
- **contacts:** opción *Contactos* del menú principal.
- **visitsPlan:** opción *Plan visitas* del menú principal.

6.5. Pruebas de *software*

Para la realización de pruebas estáticas se ha utilizado *SonarQube* ya que ofrece informes sobre arquitectura y diseño, comentarios, complejidad, duplicidad de código, estándares de codificación, evidencias de errores potenciales y pruebas unitarias. Ha resultado útil para la eliminación de código muerto, el aumento de la legibilidad del mismo y la disminución de su complejidad.

Las pruebas dinámicas se han realizado manualmente, ejecutando la aplicación y usando sus funcionalidades como si de un usuario final se tratase. Se ha evitado, en la medida de lo posible, que la persona responsable de probar una funcionalidad determinada fuera quién lo implementó, llegando a realizar las pruebas algún miembro del equipo destinado a otro proyecto.

6.6. Gestión de memoria y uso de CPU

Al desarrollar una aplicación, existen diferentes elementos y procesos que pueden hacer que la misma en ejecución tarde en visualizarse o que consuma mucha memoria en algunos casos. Para ver cuál es el coste de la aplicación se han generado diferentes gráficas intentando representar los procesos que pueden generar mayor consumo.

6.6.1. Uso de CPU y memoria en Android

Para las pruebas se ha ejecutado la aplicación en un OnePus One con la versión de Android 6.0.1

Uso de CPU

La gráfica de la Ilustración 26 es el resultado de usar todas las funcionalidades de la aplicación durante unos 15 minutos.

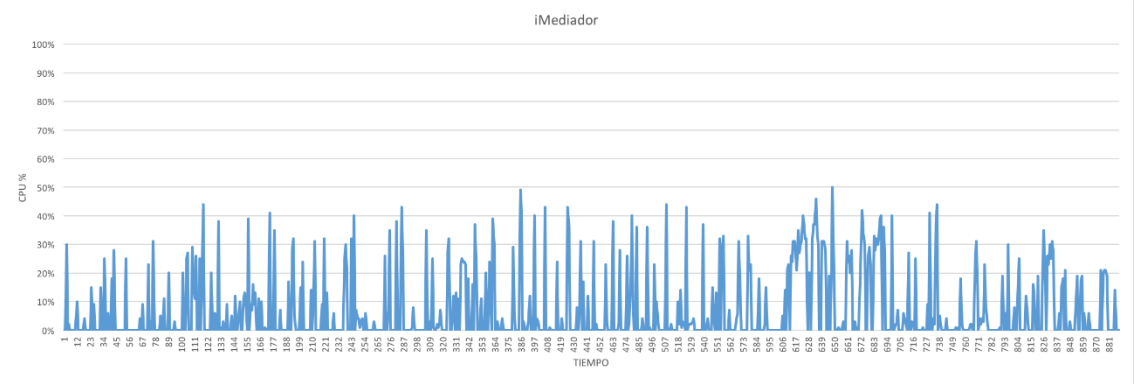


Ilustración 26: Gráfica de uso de CPU en Android

Como se puede observar, se consumen y liberan recursos a medida que se usa la aplicación, no sobrepasando nunca el 50% del uso de CPU. En la Ilustración 27 quedan mejor reflejados los puntos donde más se utiliza la CPU.

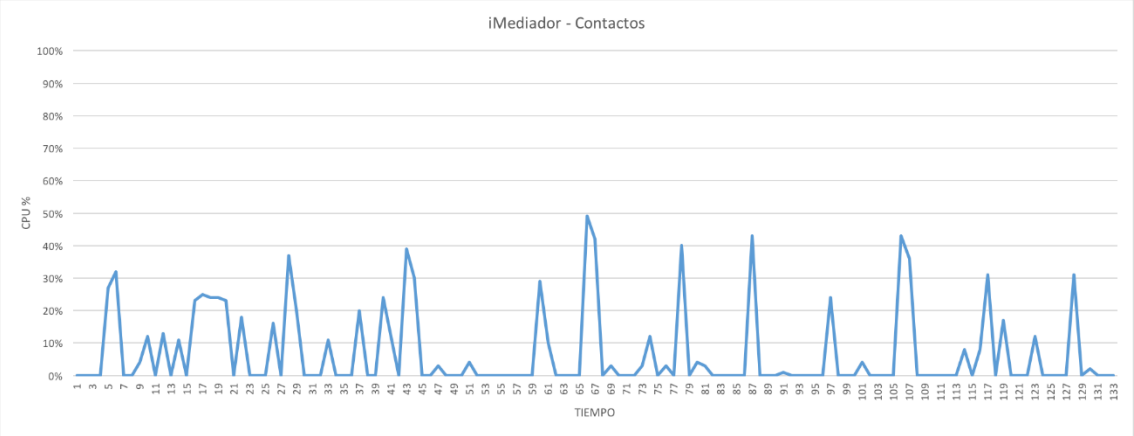


Ilustración 27: Gráfica de uso de CPU de 'Contactos' en Android

El punto máximo corresponde al acceso a la geolocalización desde el 'Detalle de contacto'. El pico de uso se debe a que se está ubicando el dispositivo al mismo tiempo que se ubica la dirección del contacto.

Uso de memoria

La gráfica de la Ilustración 28 es el resultado de usar todas las funcionalidades de la aplicación durante unos 30 minutos.

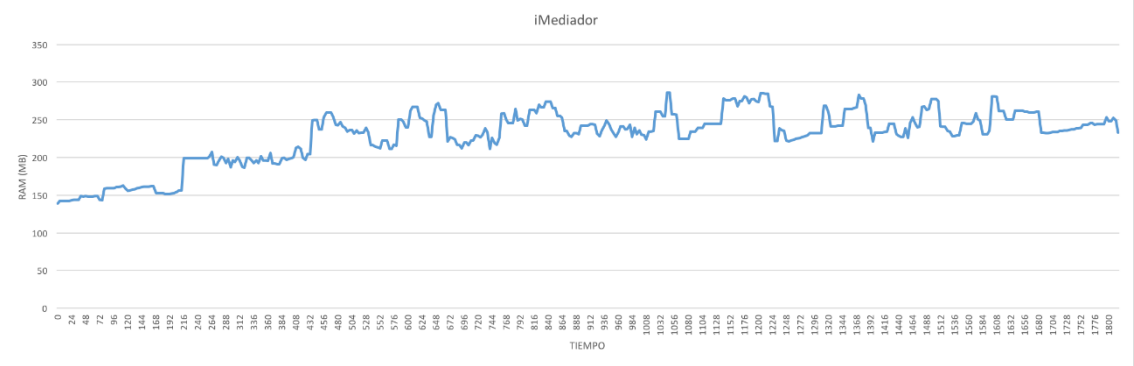


Ilustración 28: Gráfica de uso de memoria en Android

A simple vista se aprecia que se consumen y liberan recursos a medida que se usa la aplicación, siendo 286.11 MB el máximo uso de memoria, que supone el 9,31% de la memoria total (sin tener en cuenta la memoria que usa el sistema). En la Ilustración 29 queda mejor reflejado el instante donde se alcanza dicho punto.

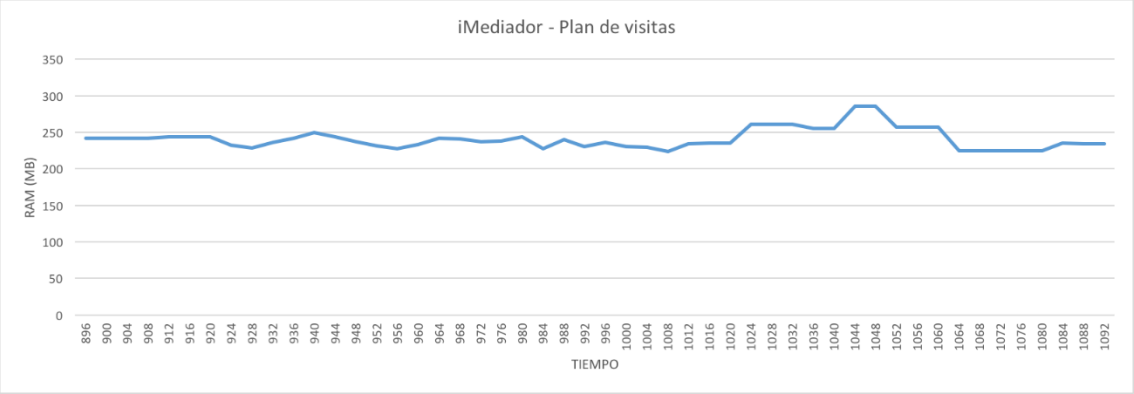


Ilustración 29: Gráfica de uso de memoria del 'Plan de visitas' en Android

Este corresponde al pintado de una ruta desde el 'Detalle de un cliente' (plan de visitas). El pico de memoria es debido a que, además del uso de recursos que conlleva cargar el mapa, la aplicación ha estado en primer plano unos 17 minutos (algunos elementos siguen cacheados).

6.6.2. Uso de memoria y CPU en iOS

Para las pruebas se ha ejecutado la aplicación en un iPhone 5 con la versión de iOS 9.3.2.

Uso de CPU

La gráfica de la Ilustración 30 es el resultado de usar todas las funcionalidades de la aplicación durante unos 20 minutos.

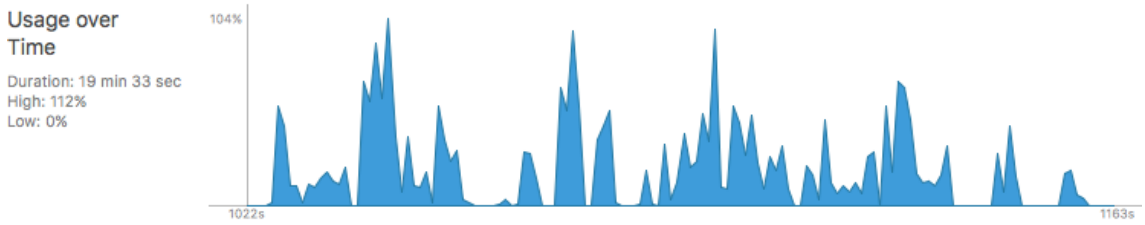


Ilustración 30: Gráfica de uso de CPU en iOS

Como se puede apreciar, se consumen y liberan recursos a medida que se usa la aplicación, no sobrepasando nunca el 112%. En iOS se considera un 100% por cada núcleo. Puesto que el iPhone 5 dispone de dos núcleos, se está usando un 112% sobre 200% (un 56% si consideramos la medida sobre 100).

El uso máximo de CPU se alcanza al intentar ubicar todas las gestiones desde el 'Listado de gestiones' (20 en el momento de la prueba) además de la ubicación actual.

Uso de memoria

La gráfica de la Ilustración 31 es el resultado de usar todas las funcionalidades de la aplicación durante unos 20 minutos.

Ilustración 31: Gráfica de uso de memoria en iOS

A simple vista se observa que se consumen y liberan recursos a medida que se usa la aplicación, siendo 188.6 MB el máximo uso de memoria, que supone el 18,42% de la memoria total (sin tener en cuenta la memoria que usa el sistema).

El uso máximo de memoria se alcanza al pintar una ruta desde el 'Detalle de un cliente' (plan de visitas). El pico de memoria se debe a que, además del uso de recursos que conlleva cargar el mapa, la aplicación ha estado en primer plano unos 18 minutos (algunos elementos siguen cacheados).

7. Conclusiones

iMediador v2.0 es una aplicación visualmente atractiva y fácil de usar. Ofrece facilidades al Colectivo de Mediadores de MAPFRE, puesto que permite realizar de manera rápida y sencilla tareas cotidianas en su entorno de trabajo, disponiendo de toda la información necesaria durante su jornada. Gracias a ella, los mediadores pueden dar de alta, modificar, cerrar o consultar el detalle de sus gestiones, así como acceder a los datos del cliente asociado a éstas o trazar rutas hasta su dirección establecida. Además, las pueden asignar a su plan de visitas como método de organización del trabajo diario y, opcionalmente, establecer la hora a la que se mantendrá la reunión con el cliente para que suene una alarma a modo de recordatorio. En cuanto a las funcionalidades relacionadas con los contactos, se permite crear, modificar o consultar la información de éstos (datos personales, vencimientos y observaciones).

A nivel tecnológico es relativamente compleja debido a la amalgama de utilidades y mecanismos que la componen, como son los servidores *MBaaS* de *Kony* y MAPFRE, la plataforma de notificaciones *MPush* y las librerías de reporte de errores y actividad (*Crashlytics* y *Adobe Mobile Services*, respectivamente).

La curva de aprendizaje de la Plataforma de Movilidad *Kony* ha sido asequible debido al grado de conocimiento del que disponen el resto de miembros del equipo.

La experiencia durante las prácticas en Okode ha sido muy gratificante. Desde el primer momento me acogieron como a un miembro más del equipo, dedicando el tiempo necesario para que me familiarizara con el conglomerado de tecnologías y buenas prácticas que utilizan, siendo algunas de ellas *MMobile, MPush, Crashlytics, Adobe Mobile Services, Jenkins, SonarQube, QUnit, Confluence, JIRA, GitHub* y *Plastic*.

Puesto que no tenía experiencia en el desarrollo de aplicaciones móviles usando *Kony* como *framework*, mi primera tarea fue seguir el *Kony Bootcamp* y realizar pequeñas aplicaciones como toma de contacto. Cuando me sentí capaz de abordar una aplicación real, me asignaron los casos de uso más concretos y sencillos que había que migrar de **iMediador v1.0** (el guardado, ordenación y edición de búsquedas favoritas; el menú de ajustes en su totalidad y la funcionalidad de añadir un contacto de la aplicación a la agenda del dispositivo).

Puntualmente realizaba pequeñas aportaciones a las tareas asignadas a mis compañeros, como el desarrollo de una librería en código nativo que permitiera ajustar la calidad, comprimir y almacenar imágenes en dispositivos Android. Ésta se utilizó para guardar imágenes desde la pantalla de parking, pero también se reutilizó en otros proyectos que se estaban llevando a cabo en Okode.

Posteriormente implementé nuevas funcionalidades propuestas para **iMediador v2.0** que permitían ordenar las gestiones del plan de visitas manualmente, obtener la dirección actual y cerrar una gestión.

En todo momento tenía en cuenta el mantenimiento de la aplicación, realizando las refactorizaciones de código pertinentes para eliminar código muerto, evitar la duplicidad del mismo o hacerlo más legible, entre otras.

Trabajos Futuros

Actualmente se está desarrollando un evolutivo de la aplicación que permitirá a los mediadores consultar la ficha de los clientes. Se podrán realizar búsquedas por datos personales y visualizar el detalle del cliente, así como la información básica de los productos que tiene contratados.

En un futuro próximo se añadirá el soporte para la orientación horizontal en *tablets*, en el que se dividirá la pantalla permitiendo tener a la izquierda el menú de la aplicación y a la derecha visualizar la opción seleccionada.

8. Bibliografía

[1] MAPFRE, «Mobile Enterprise Application Platform,» 2015. [En línea]. [Último acceso: 20 Agosto 2016].

[2] MAPFRE, «iMediador - Requisitos funcionales,» 2014.

[3] MAPFRE, «iMediador - Requisitos no funcionales,» 2014.

[4] Adobe, «Measuring and Optimizing Mobile Applications,» [En línea]. Available: https://marketing.adobe.com/developer/get-started/mobile/c-measuring-mobile-applications. [Último acceso: 20 Agosto 2016].

[5] Microsoft, «The MVVM pattern,» 10 Febrero 2012. [En línea]. Available: https://msdn.microsoft.com/en-us/library/hh848246.aspx. [Último acceso: 2016 Agosto 25].

[6] I. Sommerville, Ingeniería del Software, Madrid: Pearson Educación, S.A., 2005.

[7] Okode, «iMediador - Diseño técnico,» 2014.

[8] K. Inc., «Enterprise Mobile, Cross Platform Application Development,» 2016. [En línea]. Available: http://www.kony.com/. [Último acceso: 30 Agosto 2016].

[9] W. Chang, «Crashlytics Blog,» 21 Octubre 2014. [En línea]. Available: http://crashlytics.com/blog/introducing-fabric. [Último acceso: 20 Agosto 2016].

[10] «Jenkins,» [En línea]. Available: https://jenkins.io/. [Último acceso: 20 Agosto 2016].

[11] «SonarQube,» [En línea]. Available: http://www.sonarqube.org/. [Último acceso: 20 Agosto 2016].

[12] «El Blog de Panel Sistemas,» 11 Junio 2015. [En línea]. Available: http://blog.panel.es/index.php/softwareqa-los-7-ejes-de-la-calidad-del-codigo-fuente/. [Último acceso: 20 Agosto 2016].

[13] «QUnit,» 2016. [En línea]. Available: https://qunitjs.com/. [Último acceso: 20 Agosto 2016].

[14] Atlassian, «Confluence,» 2016. [En línea]. Available: https://es.atlassian.com/software/confluence. [Último acceso: 20 Agosto 2016].

[15] Atlassian, «JIRA,» 2016. [En línea]. Available: https://es.atlassian.com/software/jira. [Último acceso: 20 Agosto 2016].

[16] «GitHub,» 2016. [En línea]. Available: https://github.com/. [Último acceso: 20 Agosto 2016].

[17] «Plastic,» 2016. [En línea]. Available: https://www.plasticscm.com/. [Último acceso: 20 Agosto 2016].

[18] OMG, «UML,» 2016. [En línea]. Available: http://www.uml.org/. [Último acceso: 25 Agosto 2016].

[19] UPV-DSIC, *ISW: Arquitectura del software,* Valencia, 2013-2014.

[20] UPV-DSIC, *ISW: Diseño de la persistencia,* Valencia, 2013-2014.

[21] E. Freeman, E. Robson, B. Bates y K. Sierra, Head First Design Patterns, O'Reilly Media, 2004.

[22] MAPFRE, «Guía de etiquetado,» 2016.

[23] «Wikipedia,» 2 Mayo 2016. [En línea]. Available: https://es.wikipedia.org/wiki/Backend_as_a_service. [Último acceso: 24 Agosto 2016].

[24] «Wikipedia,» 29 Mayo 2016. [En línea]. Available: https://es.wikipedia.org/wiki/SQLite. [Último acceso: 25 Agosto 2016].

Apéndices

A. Manual de usuario

En este apartado se explican las distintas pantallas de la aplicación agrupadas por funcionalidad. Para evitar diagramas excesivamente complejos se ha optado por disgregar en varios apartados a costa de repetir algunas imágenes.

Login

En la Ilustración 32 se muestran las primeras pantallas que veremos al arrancar la aplicación. En caso de ser la primera vez que se ejecute, nos solicitará que nos identifiquemos mediante NUUMA y contraseña y, posteriormente, nos pedirá un PIN para poder iniciar sesión más fácilmente en los siguientes inicios de la aplicación. Después de autenticarnos con uno de los dos métodos, veremos el menú principal con todas las opciones disponibles.

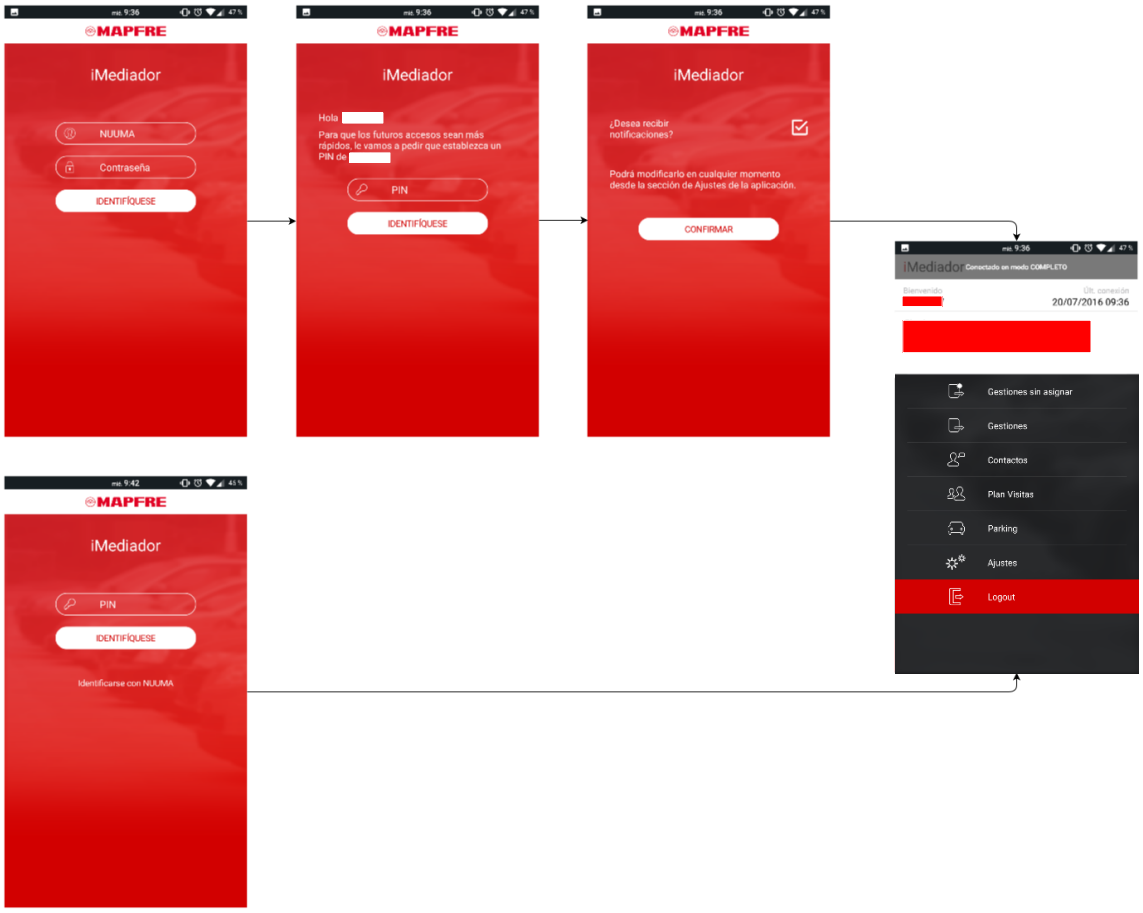


Ilustración 32: Login

Menú principal

En la Ilustración 33 se pueden observar las distintas opciones a las que tiene acceso el usuario, así como ciertos datos que se han ocultado para no comprometer su información personal.

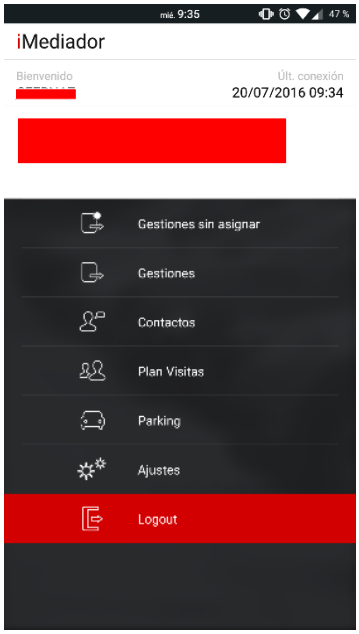


Ilustración 33: Menú principal

Gestiones sin asignar

La primera pantalla que vemos tras pulsar sobre la opción del menú "Gestiones sin asignar" es una clasificación según el tipo de gestiones que queramos mostrar (Ilustración 34).

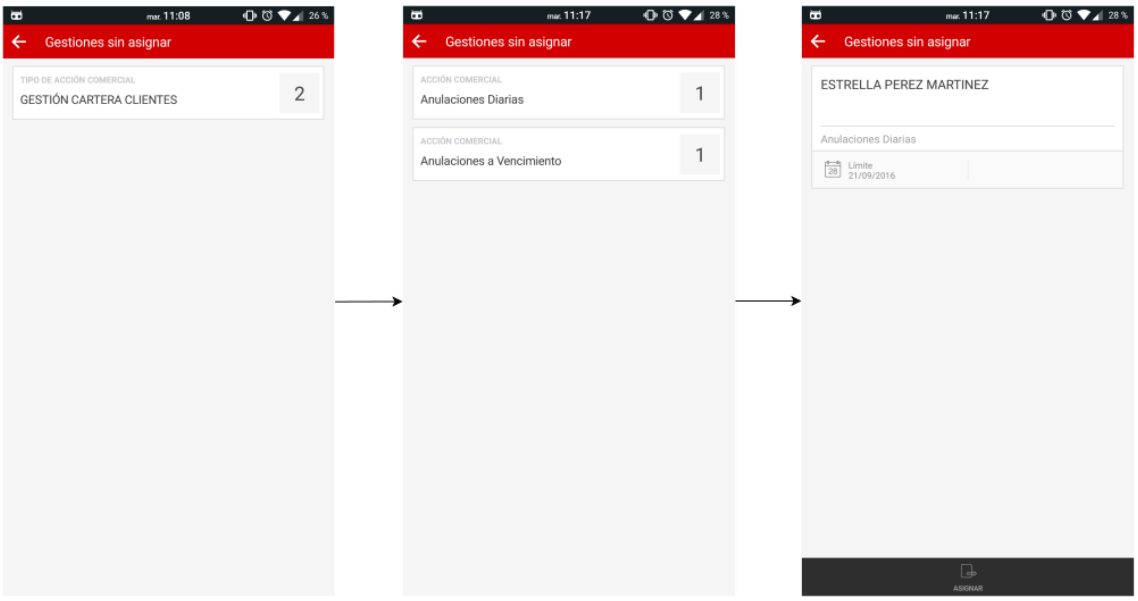


Ilustración 34: Gestiones sin asignar

Listado de gestiones

Después de seleccionar el tipo de gestiones, nos aparece un listado como el de la Ilustración 35, donde posteriormente podemos ver el detalle de la gestión, asignarla a un miembro de nuestro equipo o mostrar su ubicación en el mapa interactivo.

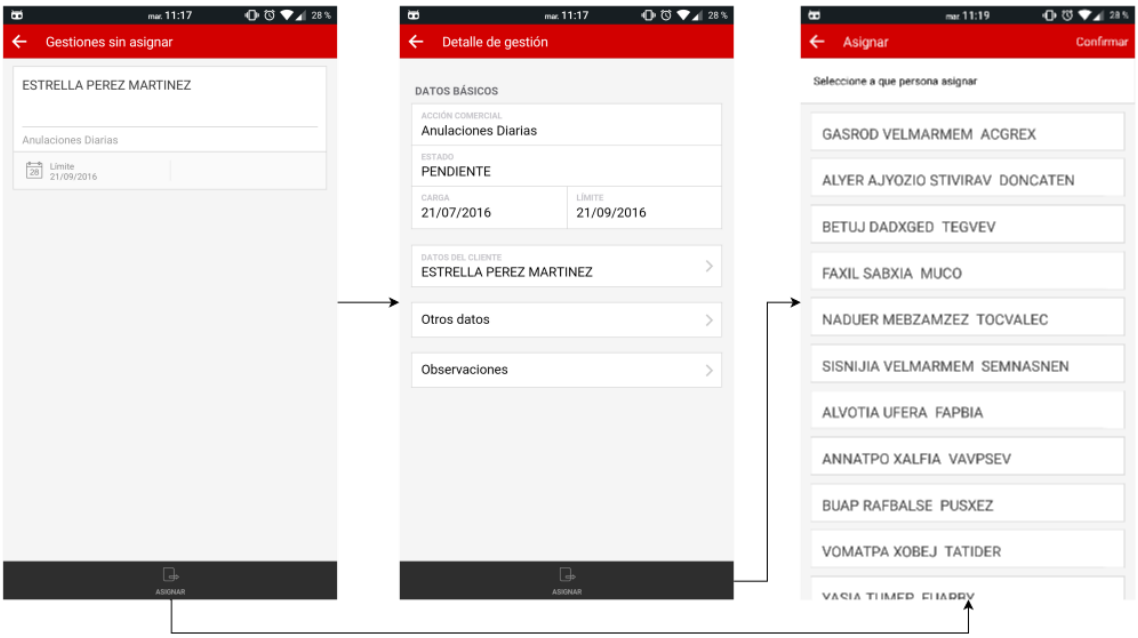


Ilustración 35: Gestiones sin asignar - Listado de gestiones

Detalle de gestión

Desde el detalle de gestión es posible acceder a los datos del cliente y a otra información relevante relacionada con la propia gestión. Al igual que desde el listado, se puede asignar la gestión a otro miembro del equipo o mostrar su ubicación. Esto queda reflejado en la Ilustración 36.

Gestiones

En la primera pantalla que vemos tras pulsar sobre la opción del menú “Gestiones” (Ilustración 37) se nos ofrece la posibilidad introducir los criterios de búsqueda manualmente o mostrar la pantalla de favoritos, donde se muestran los criterios de búsqueda almacenados. Desde cualquiera de ellas, se navega al listado de gestiones, que se explica detalladamente continuación, o navegar a la pantalla de “Nueva gestión”.

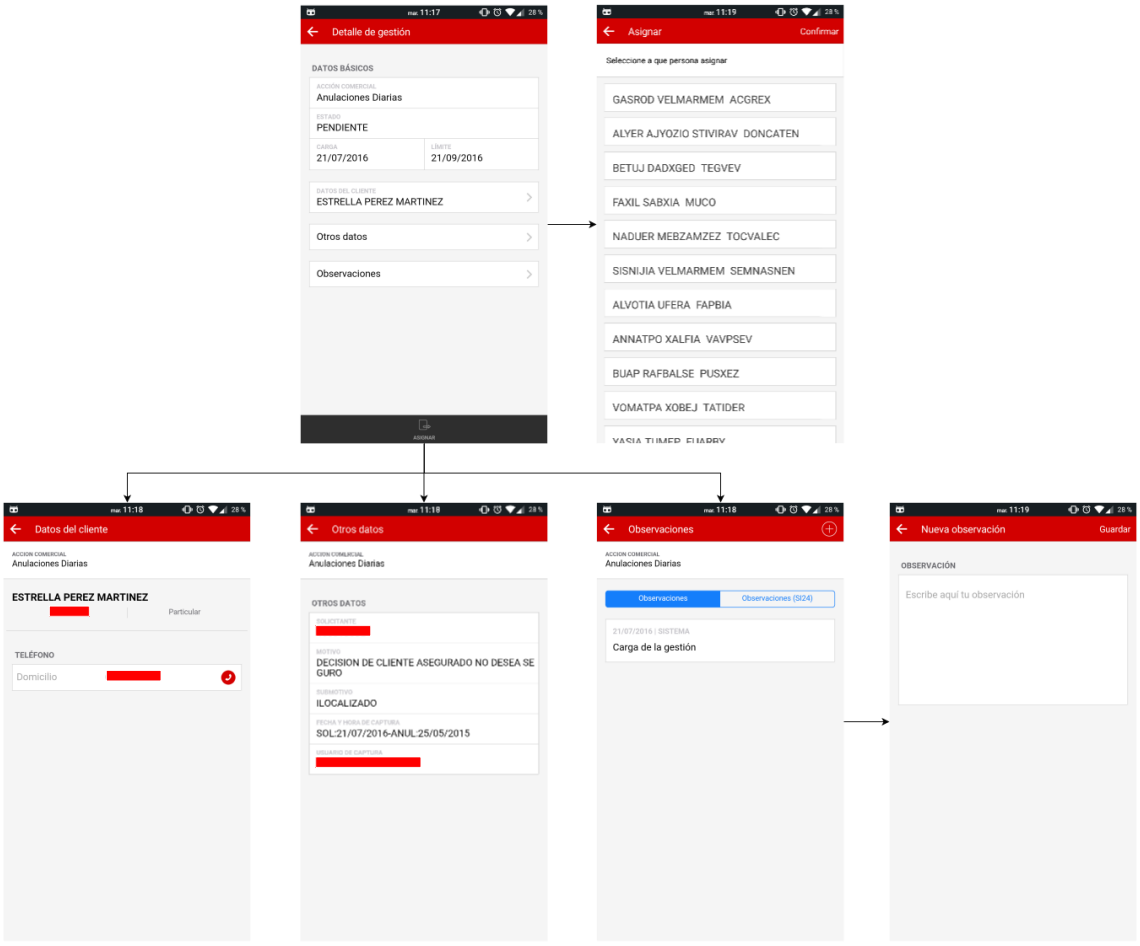


Ilustración 36: Gestiones sin asigna - Detalle de gestión

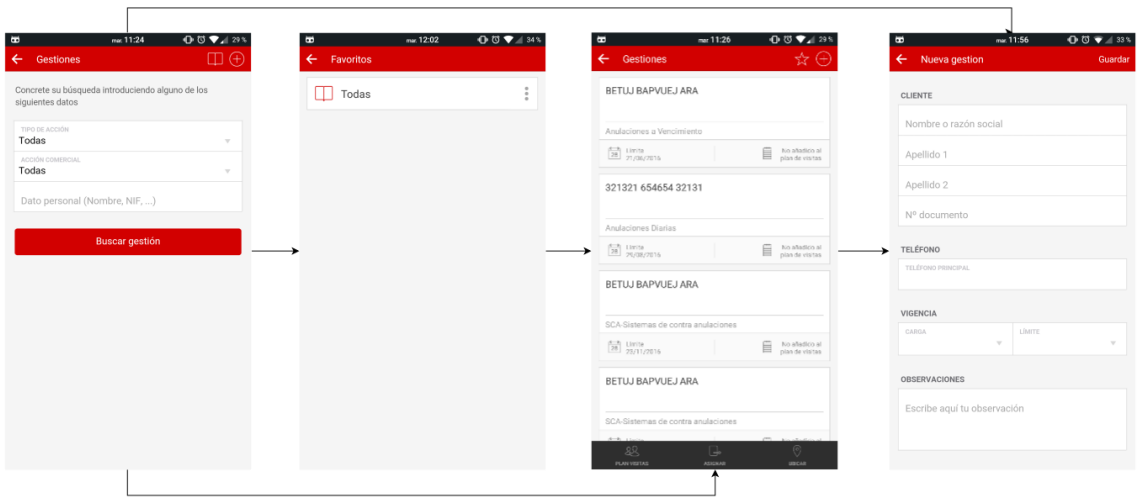


Ilustración 37: Gestiones

Listado de gestiones

Una vez en el listado de gestiones, se puede guardar el criterio por el que se ha buscado como favorito, acceder al detalle de una gestión o crear una, enviar una o más gestiones al plan de visitas, asignar una o más gestiones a otro miembro del equipo o mostrar la ubicación de todas ellas en el mapa interactivo. Todas ellas quedan reflejadas en la Ilustración 38.

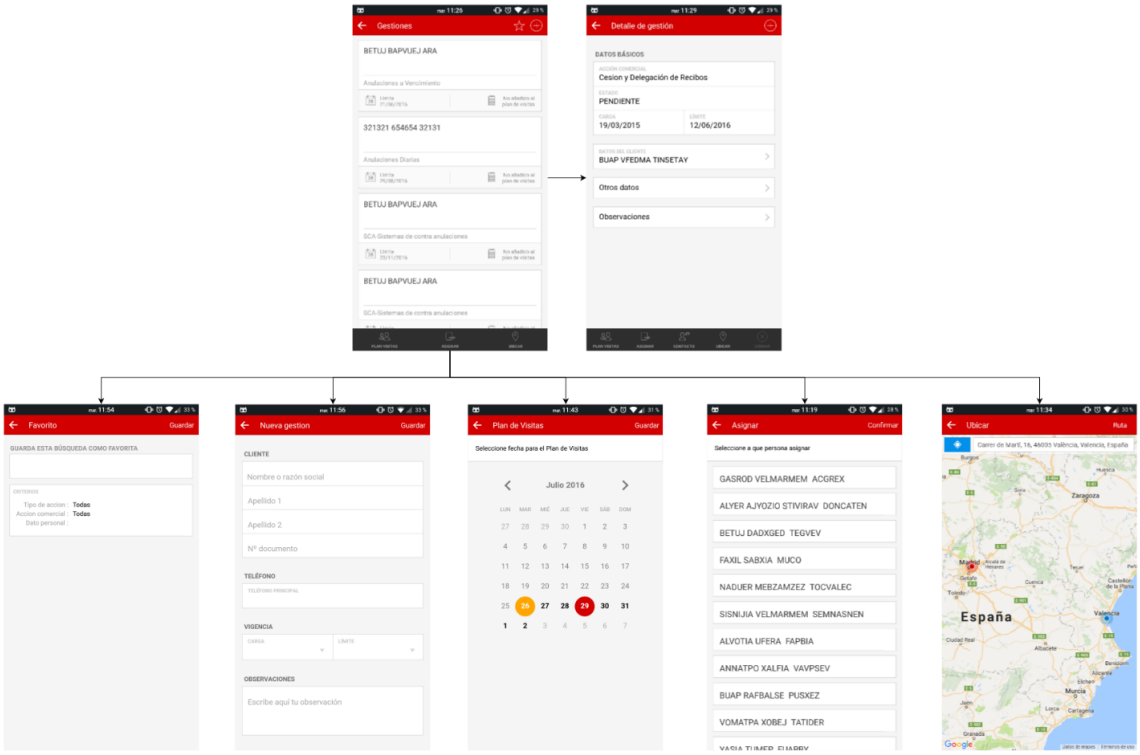


Ilustración 38: Listado de gestiones

Detalle de gestión

Desde el detalle de gestión es posible crear una gestión propia (esto es, crearla y asignarla al usuario actual), acceder a los datos del cliente o del contacto y a otra información relevante relacionada con la propia gestión, enviarla al plan de visitas, asignársela a otro miembro del equipo o mostrar su ubicación en el mapa interactivo como se puede observar en la Ilustración 39.

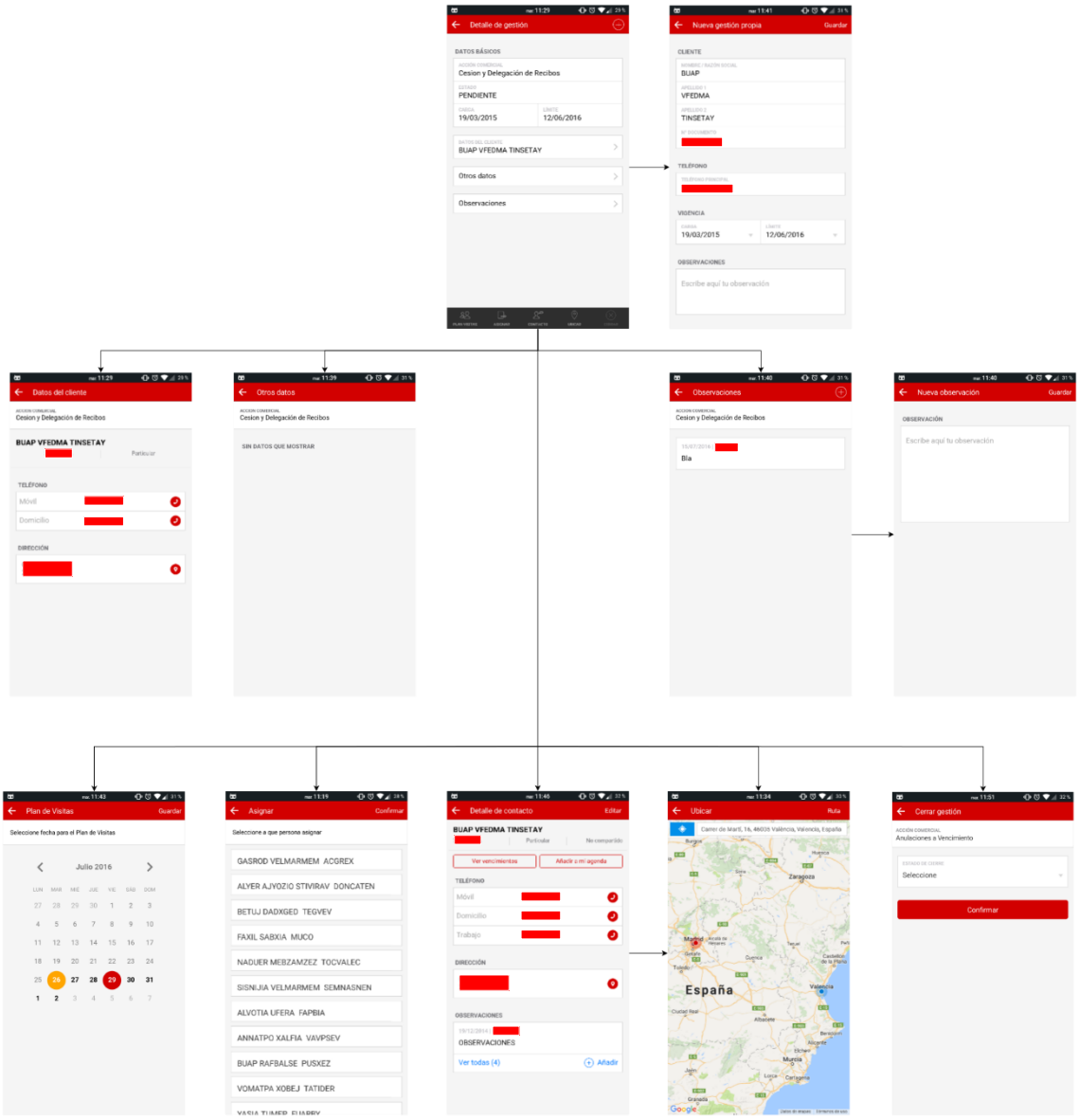


Ilustración 39: Detalle de gestión

En la Ilustración 40 podemos ver todas la pantallas alcanzables desde el detalle de un contacto, es decir, editar un contacto y consultar, crear o editar sus vencimientos u observaciones.

Detalle de contacto

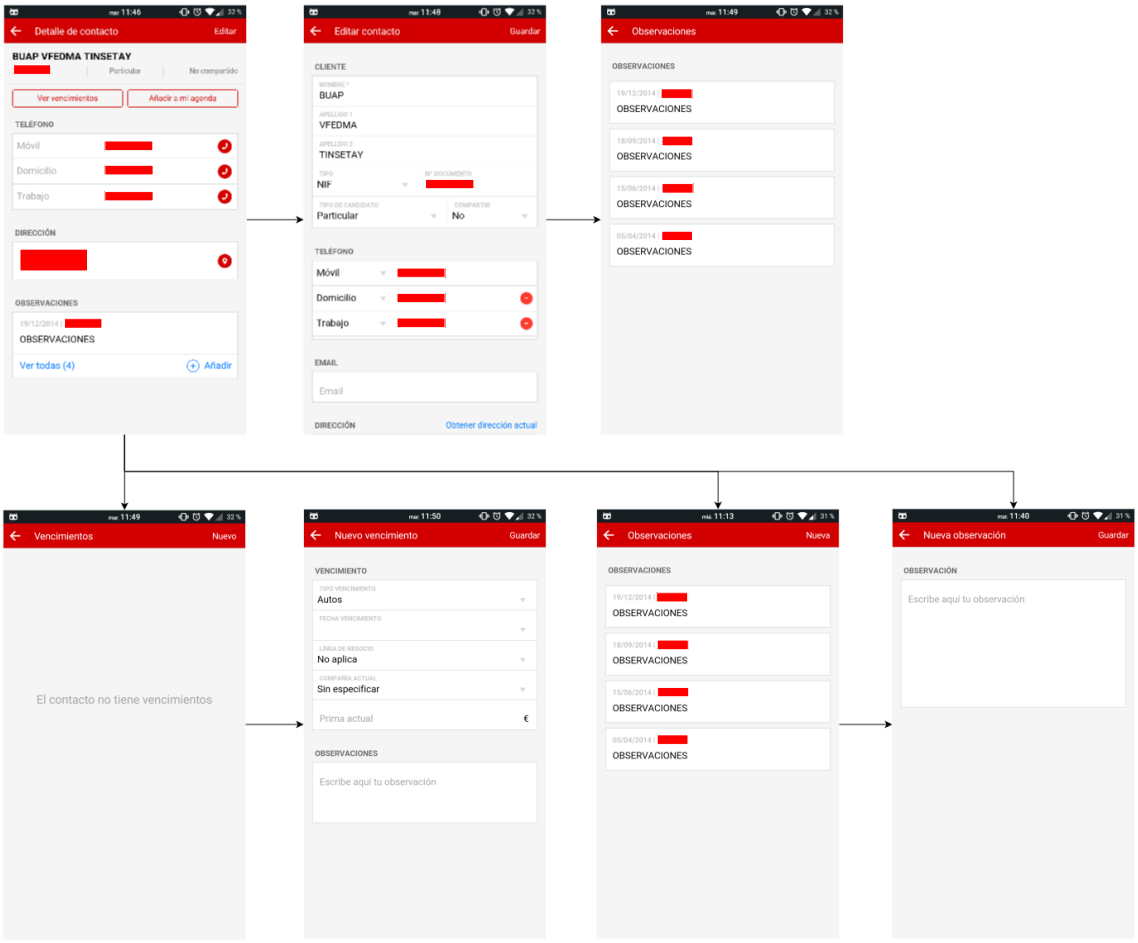


Ilustración 40: Detalle de contacto

Contactos

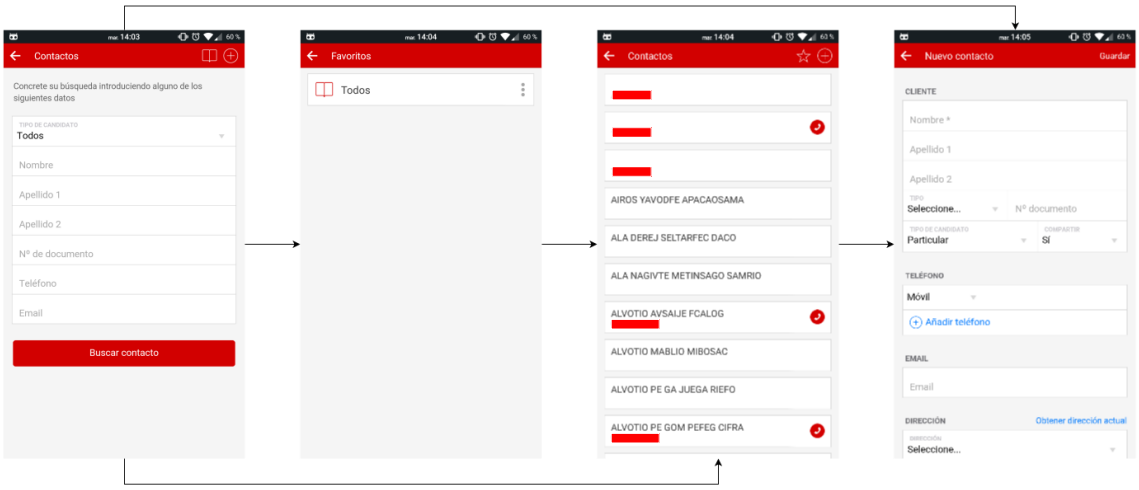


Ilustración 41: Contactos

Tras pulsar sobre la opción del menú “Contactos” se muestra la pantalla de búsqueda de criterios, donde también podemos navegar a los favoritos. Desde cualquiera de ellas,

se navega al listado de contactos, como se explica a continuación, o a la pantalla de "Nuevo contacto". Todo ello puede observarse en la Ilustración 41.

Listado de Contactos

Una vez en el listado de contactos, se puede guardar el criterio por el que se ha buscado como favorito, acceder al detalle de un contacto o crear uno, como puede observarse en la Ilustración 42.

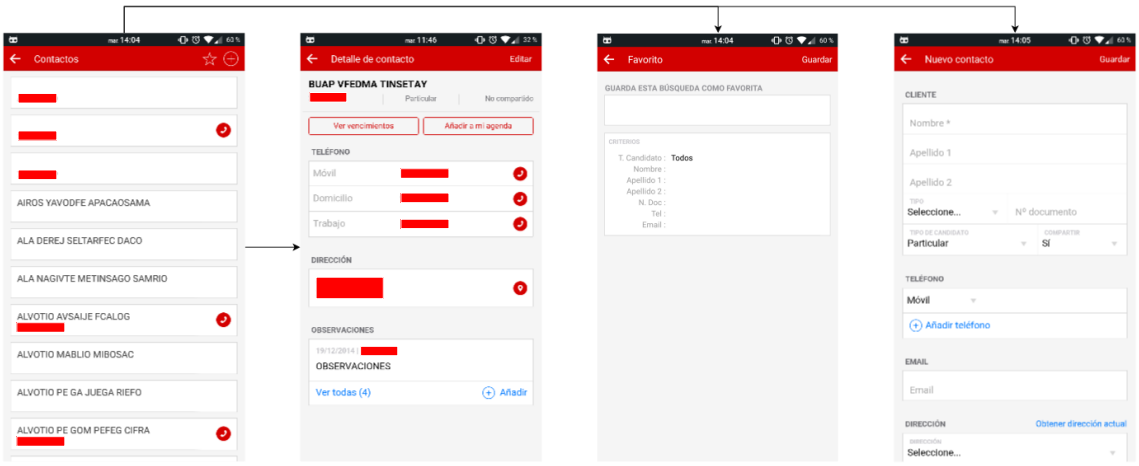


Ilustración 42: Listado de contactos

En la Ilustración 43 podemos ver todas la pantallas alcanzables desde el detalle de un contacto, es decir, editar un contacto y consultar, crear o editar sus vencimientos u observaciones.

Detalle de contacto

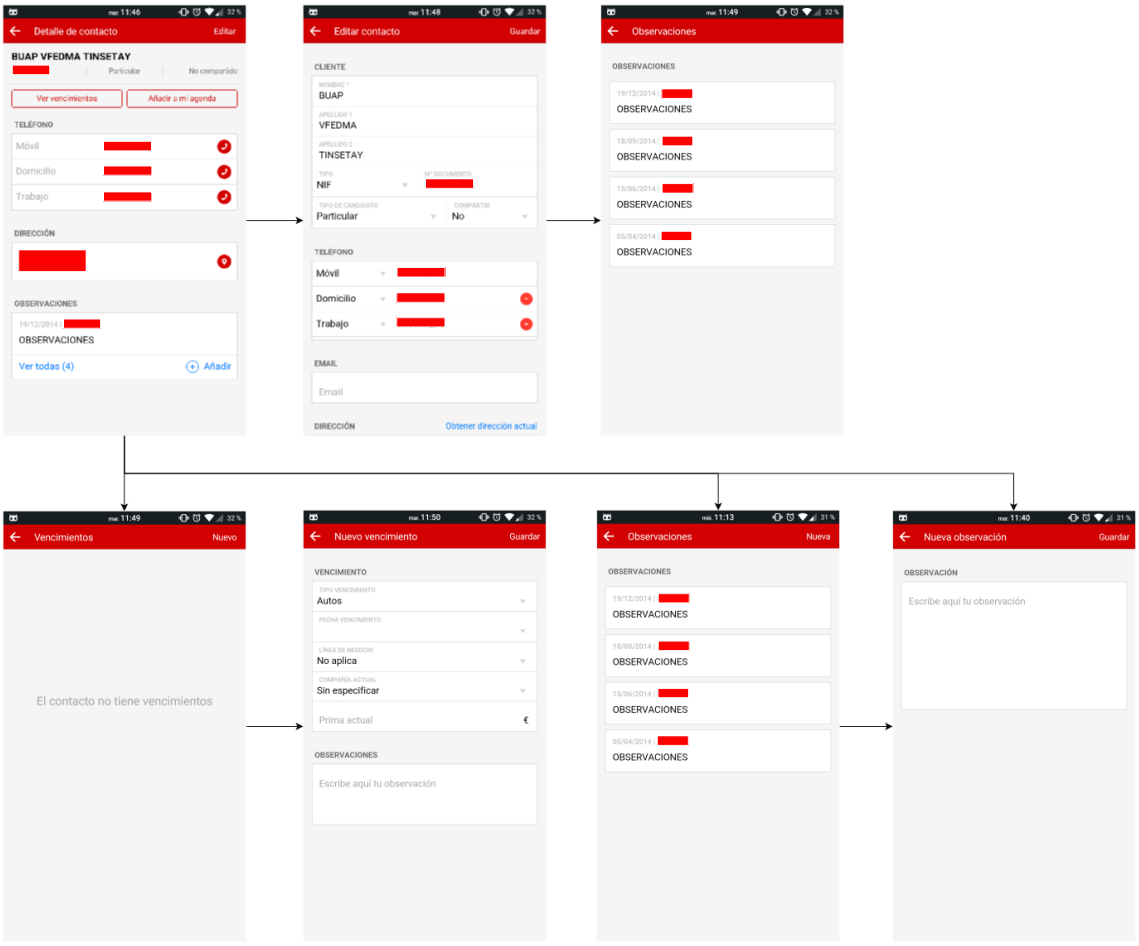


Ilustración 43: Detalle de contacto

Plan Visitas

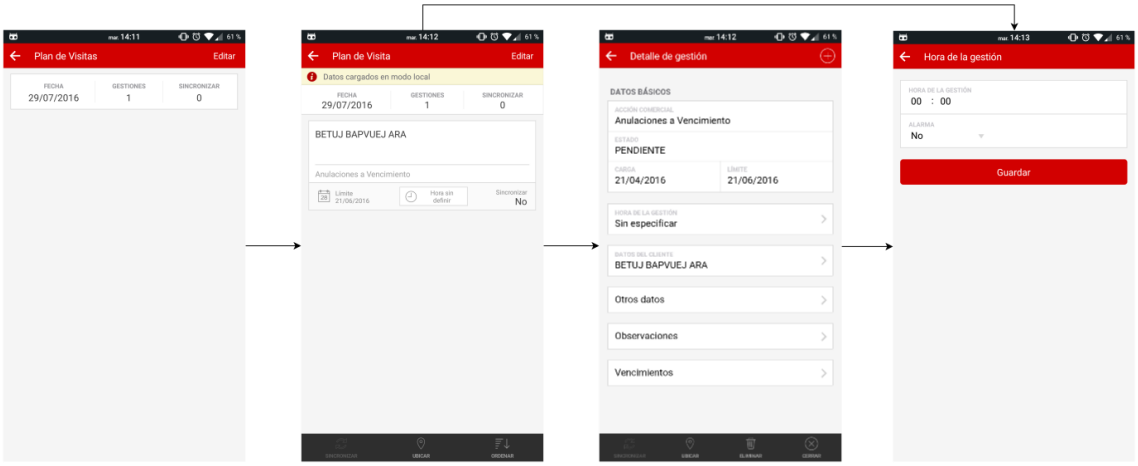


Ilustración 44: Plan de visitas

Al pulsar sobre la opción “Plan Visitas”, veremos un listado de gestiones planificadas organizadas por fecha. Al pulsar sobre un elemento de la lista, se mostrará un listado de

gestiones planificadas para la fecha seleccionada. Desde éste, se podrá navegar al detalle de la gestión o establecer una hora para la cita con el cliente. Esto queda reflejado en la Ilustración 44.

Detalle de gestión

Desde el detalle de gestión es posible crear una gestión, establecer la hora a la que se ha concertado la cita con el cliente; acceder a los datos del mismo; consultar, crear o modificar las observaciones, los vencimientos y otra información relevante relacionada con la propia gestión; o mostrar su ubicación en el mapa interactivo como se puede observar en la Ilustración 45.

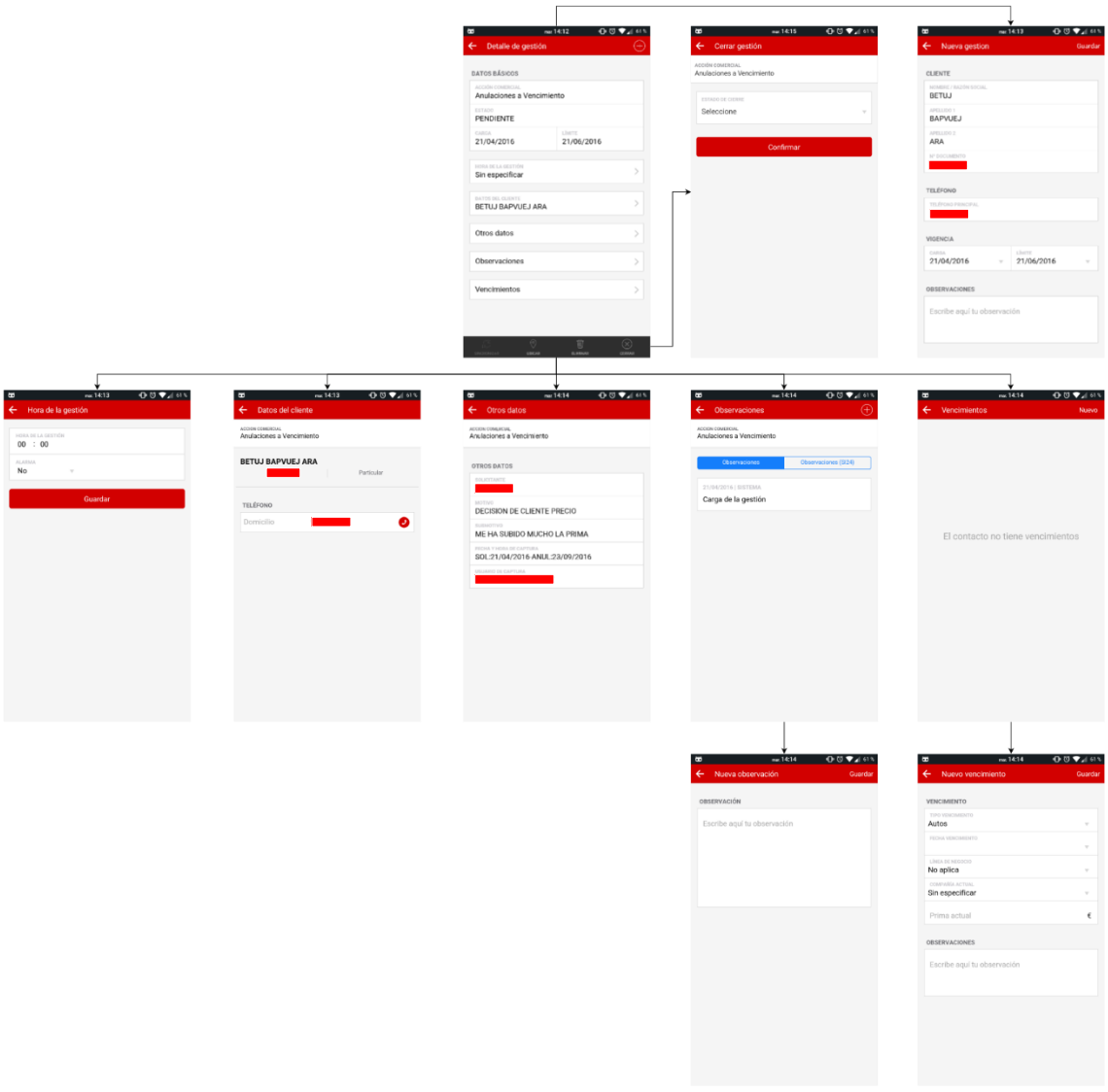


Ilustración 45: Detalle de gestión

Ubicación

La pantalla de la Ilustración 46 es común para las opciones de menú “Gestiones sin asignar”, “Gestiones”, “Contactos” y “Plan Visitas”. Se puede acceder desde el listado de

gestiones o el detalle de una en el caso de “Gestiones sin asignar”; desde el listado de gestiones, los datos de un cliente o el detalle de un contacto en el caso de “Gestiones”; desde el detalle de un contacto en el caso de “Contactos”; o desde el listado de planes o datos de un cliente en el caso de “Plan Visitas”.

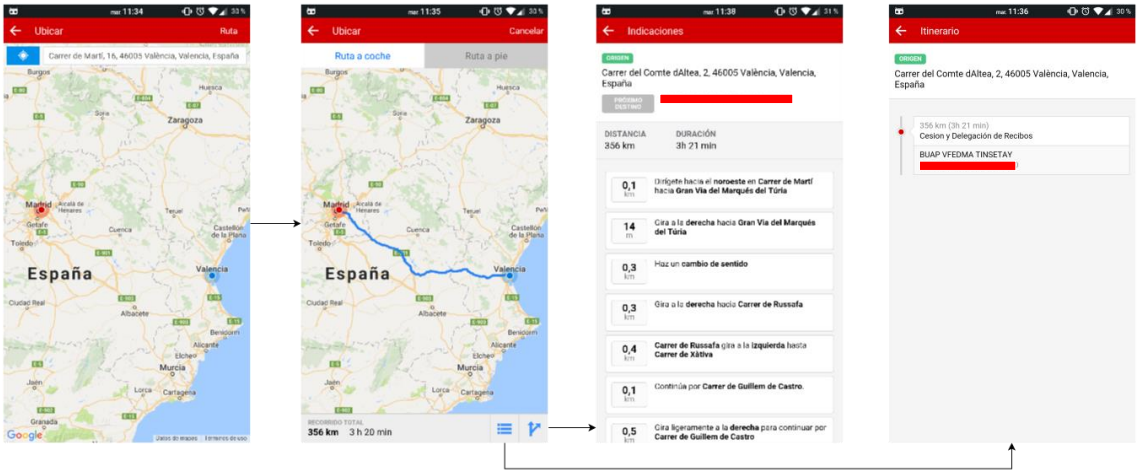


Ilustración 46: Ubicación

Parking

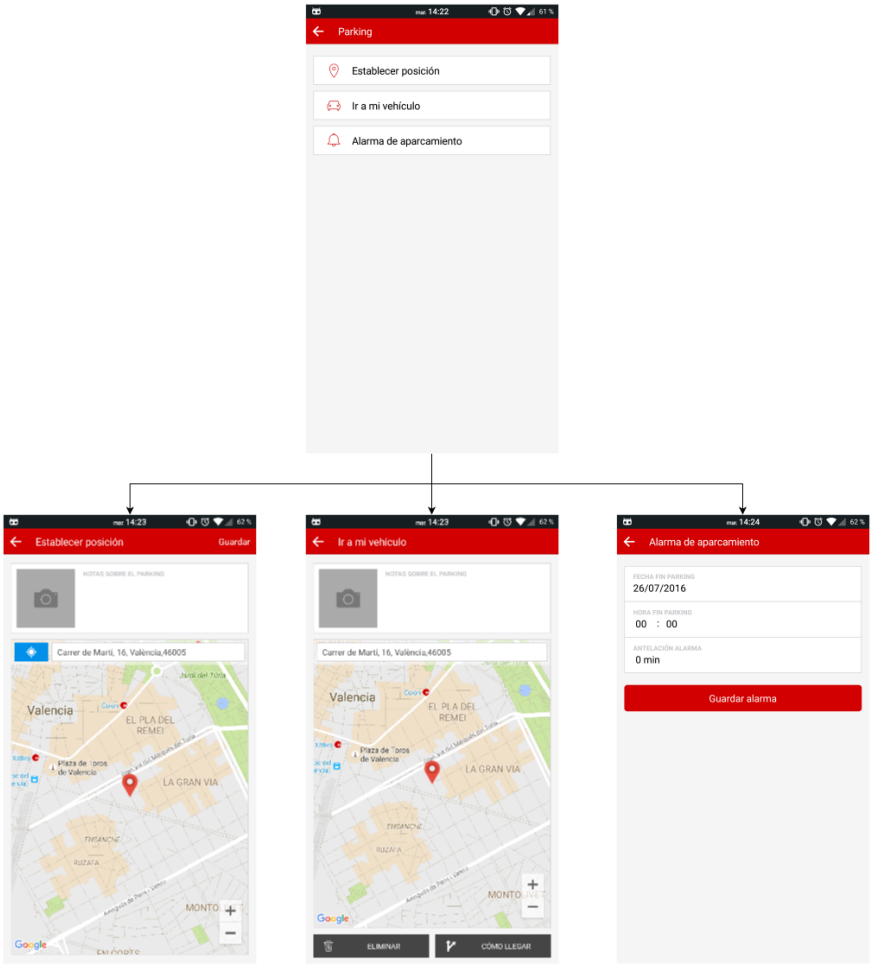


Ilustración 47: Parking

Al pulsar sobre la opción del menú “Parking”, navegamos a una pantalla donde se nos permite elegir entre tres opciones: establecer posición, ir al vehículo y alarma de aparcamiento. En la primera, es posible guardar la posición actual, un texto personalizado e incluso una imagen. En la segunda, se mostrarían los datos que se han almacenado previamente. Por último, la alarma nos serviría en zona de estacionamiento regulado para no permanecer más tiempo del permitido. Todo ello queda reflejado en la Ilustración 47.

Ajustes

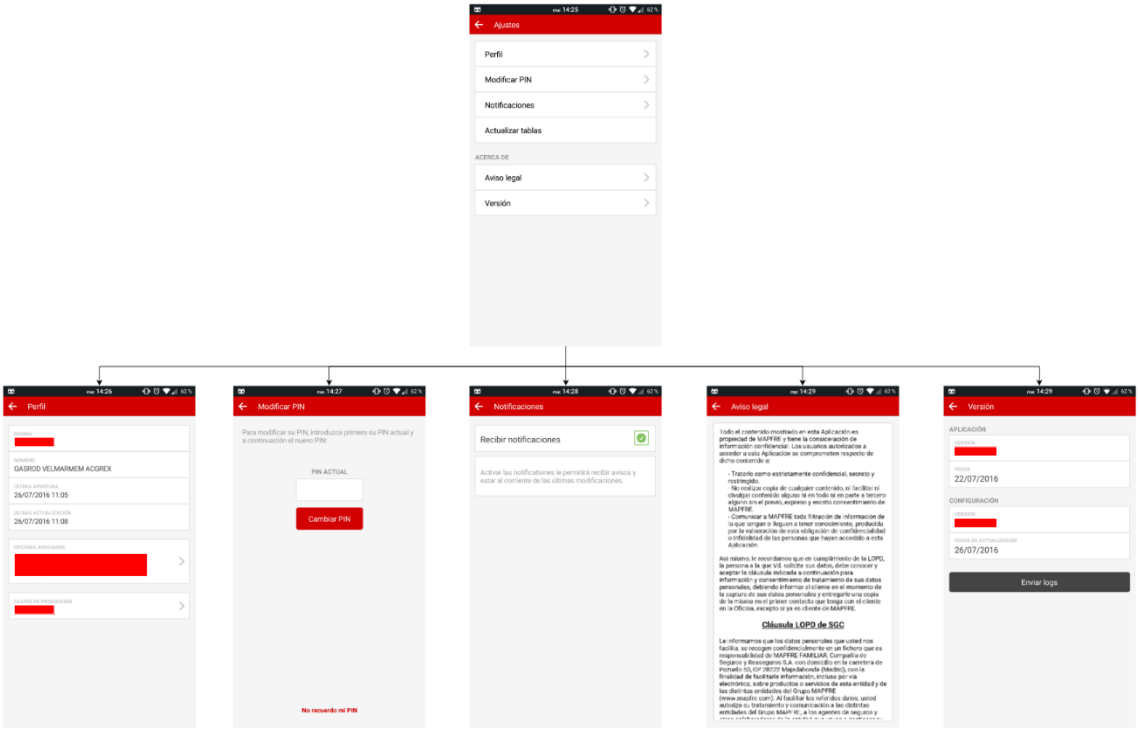


Ilustración 48: Ajustes

La última opción menú, “Ajustes”, permite mostrar información del perfil del usuario, modificar su pin, activar o desactivar las notificaciones, visualizar el aviso legal, mostrar información relacionada con la versión de la aplicación y enviar el registro de *logs*. Todas estas opciones se pueden ver en la Ilustración 48.

La información del perfil del usuario que podemos consultar, se muestra en la Ilustración 49. Esto es, su nombre y apellidos, oficina asociada y miembros de su mismo equipo.

Por último, en la Ilustración 50, se muestra el proceso de cambio de PIN. Se puede introducir el antiguo PIN o bien el NUUMA y la contraseña y, posteriormente, el nuevo PIN.

Perfil

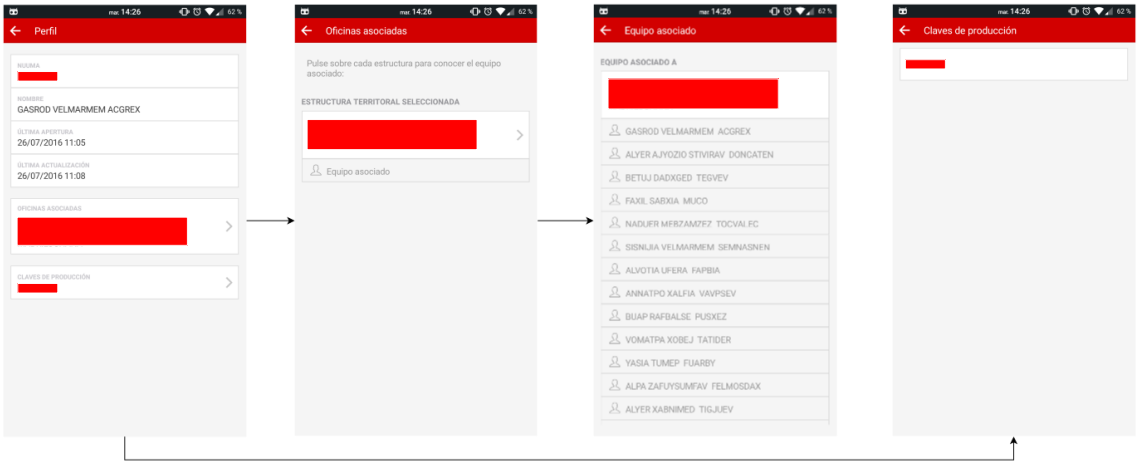


Ilustración 49: Perfil

Modificar PIN

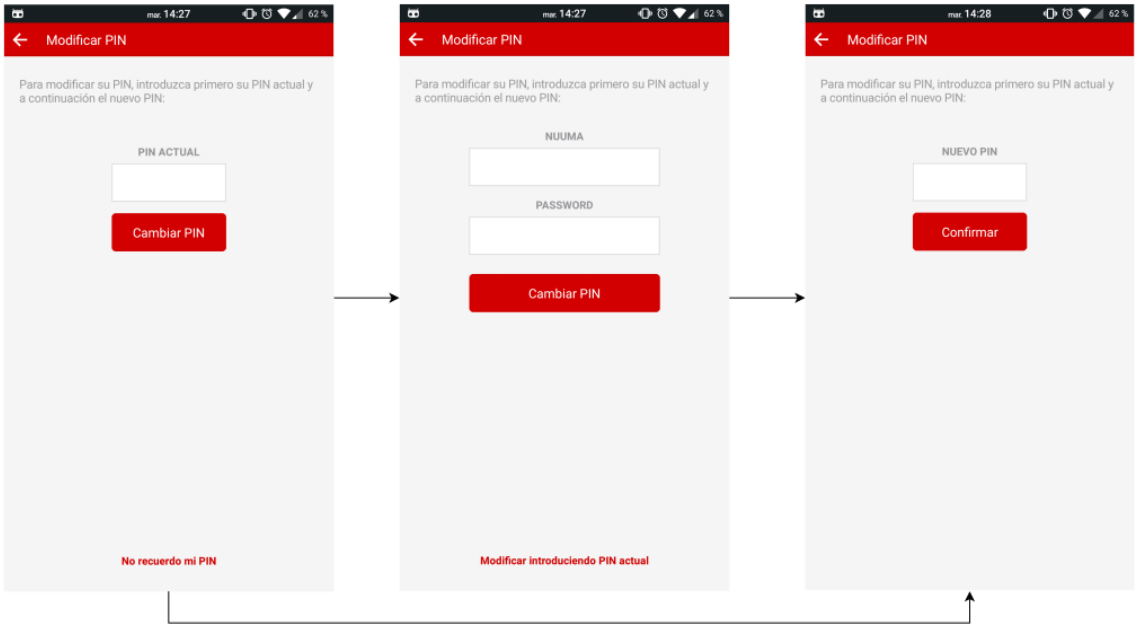


Ilustración 50: Modificar PIN