Universidad Zaragoza
1542

Trabajo Fin de Máster

Análisis y comprensión del funcionamiento de las tecnologías para la creación de aplicaciones móviles multiplataforma: propuesta de uso dentro del proyecto aGROSLab

Autor

Jorge Sanz Alcaine

Director

Juan López de Larrínzar

Ponente

Francisco Javier Zarazaga Soria

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2020

Resumen

Las tecnologías para el desarrollo de aplicaciones móviles multiplataforma han avanzado exponencialmente en los últimos tiempos y se han convertido en una opción perfectamente viable para la construcción de soluciones industriales incluso en los entornos más exigentes. De este modo, en los últimos años han surgido algunos frameworks de este tipo que han ganado notable popularidad en las empresas debido al ahorro que suponen en el tiempo de desarrollo. Cada uno de estos framworks tienen alguna característica que la distingue el resto. Por ejemplo, Xamarin es especialmente útil para aplicaciones de alto rendimiento, React Native cuenta con una gran comunidad de usuarios, o Flutter reduce aún más el tiempo de desarrollo.

Dentro de los modelos de desarrollo multiplataforma, en los últimos años se ha popularizado uno que consiste en desarrollar progressive web apps en lugar de aplicaciones nativas. Las progressive web apps son aplicaciones con estilo nativo, funcionalidades nativas y soporte offline, pero que no son aplicaciones nativas, sino que son aplicaciones web. Son una tecnología bastante nueva y, recientemente, tanto Google con Apple han empezado a dar soporte en sus navegadores. Sin embargo, las restricciones en las tiendas de Google y Apple siguen siendo bastante estrictas comparadas con las aplicaciones nativas. El planteamiento más habitual es el que los framworks permitan generar aplicaciones nativas mediante Webviews. Éstos son navegadores embebidos en una aplicación. Año tras año los webviews incluyen más funcionalidad, pero actualmente siguen sin cubrir todas las funcionalidades del dispositivo. Es por ello que en muchas soluciones se hace uso de plugins que ofrecen un API javascript para acceder a la funcionalidad nativa del dispositivo.

Dentro de los frameworks existentes, para este proyecto se ha seleccionado Ionic para desarrollar los complementos de aplicación móvil para el proyecto aGROSLab. Concretamente se ha puesto el foco en el cuaderno de explotación de aGROSLab que surge como respuesta al real decreto 1311/2012, que exige a los agricultores el asesoramiento de un técnico acreditado al utilizar productos fitosanitarios. Por ello, se han elaborado dos aplicaciones. La aplicación del asesor, que permite a los asesores realizar prescripciones a sus agricultores asociados y la aplicación del agricultor, que permite a los agricultores registrar los tratamientos realizados. Existen otras tecnologías que encajan perfectamente con Ionic para el desarrollo multiplataforma y que también se han utilizado en este proyecto.

El desarrollo del proyecto se ha realizado entre los meses de mayo y noviembre, y se ha seguido una adaptación de metodologías ágiles para un trabajo de TFM.

Índice

1 Introducción

En este documento se describe el proceso de desarrollo y los resultados del estudio en el trabajo titulado “Análisis y comprensión del funcionamiento de las tecnologías para la creación de aplicaciones móviles multiplataforma: propuesta de uso dentro del proyecto aGROSLab”

1.1 Descripción del proyecto aGROSLab

aGROSLab es una iniciativa liderada por las empresas 7eData y GeoSpatiumLab, que cuenta con la colaboración de otras entidades entre las que se incluyen diversos grupos de investigación de la Universidad de Zaragoza. aGROSLab se concibe como una plataforma digital que facilita la Asociación Colaborativa de los Actores del Sector Agrario. Para ello trabajo sobre la premisa de la integración entre los Productores Agrarios y la Industria Agroalimentaria como un factor clave del futuro en el que los principales actores van a ser:

- Asesores de Explotaciones Agrarias que deben actuar para evolucionar el sector y hacerlo eficiente, eficaz y sostenible. Para ello deberán apoyarse en las nuevas tecnologías y de su aplicación en el lugar y en la medida adecuada.
- Agricultores que van a buscar una producción eficiente y sostenible aplicando las técnicas más avanzadas.
- Agroindustria que, junto a productores y asesores, debe trabajar adecuando la producción a la demanda del mercado y generando valor añadido.
- Otros actores que agrupan a diferentes colectivos de productores agrarios, defendiendo sus intereses, generando valor añadido, gestionando recursos escasos, facilitando el intercambio de conocimiento y prestando asistencia técnica y formación.

aGROSLab es la plataforma que permite interactuar a todos los agentes, facilitando la incorporación de las nuevas tecnologías al sector e integrando los datos producidos en un único repositorio que facilitará la generación de información para la toma de decisiones. En la actualidad, aGROSLab mantiene dos líneas de producto: aGROSLab – Cuaderno de Explotación y aGROSLab – GO.

aGROSLab – Cuaderno de Explotación es una plataforma web desarrollada para dar soporte al Registro de la Aplicación de Productos Fitosanitarios en las Explotaciones Agrícolas, de acuerdo a lo establecido en Real Decreto 1311/2012 [1], de 14 de septiembre, por el que se establece el marco de actuación para conseguir un uso sostenible de los productos fitosanitarios. aGROSLab – Cuaderno de Explotación ha sido diseñado para facilitar, tanto a agricultores como a Asesores Profesionales, la elaboración del Cuaderno de Explotación. Se concibe como una herramienta especialmente adecuada para Cooperativas Agrarias, facilitando la confección de los cuadernos de explotación de sus socios, permitiendo la carga de datos (productos fitosanitarios, producción, …) desde sus aplicaciones de gestión.

aGROSLab – GO es una línea de producto orientada al desarrollo de Servicios Basados en la Localización en entornos agrarios. Su objetivo es complementar a aGROSLab – Cuaderno de Explotación llevando funcionalidades de éste a dispositivos móviles siempre que la ubicación y georreferenciación resulte necesaria. Por otro lado, se abordan trabajos de I+D+i para ser capaces de ofrecer soluciones de movilidad en entornos de comunicaciones degradadas (con bajas velocidades de comunicación móvil, o incluso sin comunicación móvil). En este proyecto se utiliza Ionic para el desarrollo de aGROSLab – GO en dispositivos IOS y Android.

1.2 GeoSpatiumLab

GeoSpatiumLab [2] es una empresa de base tecnológica, Spin-off de la Universidad de Zaragoza, especializada en el tratamiento digital de la información geoespacial y georreferenciada y sus ámbitos de aplicación. La empresa se creó en enero de 2007 para transferir la tecnología del Grupo de Sistemas de Información Avanzados (IAAA) de la Universidad de Zaragoza.

Desde su fundación, GeoSpatiumLab colabora con organizaciones de estandarización y cuenta con alianzas tecnológicas para I+D con empresas y centros públicos de investigación. La empresa ha participado en numeroso proyectos nacionales y europeos de investigación e innovación, además de mantener una política activa de publicación de resultados en congresos y revistas especializadas.

Los ámbitos tecnológicos en los que trabaja GeoSpatiumLab son, principalmente, los Sistemas de Información Geográfica, Servicios Basados en la Localización, Servicios y aplicaciones Web, y Aplicaciones para dispositivos móviles. El sector agrario es uno de principales ámbitos de trabajo en GeoSpatiumLab.

1.3 Objetivos

El proyecto tiene un doble objetivo:

- Comprender el funcionamiento del stack tecnológico vinculado a la creación de aplicaciones móviles multiplataforma y proponer un recorrido tecnológico para su uso. Analizar las alternativas, los condicionantes para multiplataforma y lo que ofrece este tipo de desarrollo.
- Utilizar las tecnologías analizadas para desarrollar una aplicación industrial para la gestión de prescripciones fitosanitarias. La aplicación deberá ser capaz de visualizar e interactuar con mapas, funcionar sin cobertura de datos y acceder a los recursos del dispositivo.

La aplicación está destinada principalmente a agricultores, que por lo general no tienen conocimientos informáticos. Por tanto, es fundamental que la aplicación sea sencilla y que el agricultor no se quede atascado en ningún formulario.

1.4 Estado del arte

Para comprender la importancia que tienen los frameworks de desarrollo multiplaforma es necesario analizar primero el entorno en el que se encuentran y las capacidades que tienen actualmente. El mercado de plataformas móviles está dividido principalmente entre Android de Google e IOS de Apple. Ambas son incompatibles a nivel de aplicación, por lo que a la hora de desarrollar aplicaciones que se ejecuten tanto en Android como en IOS existen varios enfoques [3].

- Desarrollar una aplicación web basada en HTML5, CSS y JavaScript. También llamadas Progressive Web Apps.
- Desarrollar aplicaciones nativas para cada una de las plataformas. Típicamente suele suponer multiplicar el esfuerzo por el número de plataformas a desplegar.
- Utilizar un framework de desarrollo híbrido o multiplataforma.

Utilizar un framework de desarrollo multiplataforma suele ser la opción preferida por empresas que no tengan los recursos para emplear un equipo de desarrollo especializado para cada plataforma. Además, la idea de tener un único código base facilita bastante el desarrollo y el mantenimiento de la aplicación.

Existe una gran variedad de frameworks para el desarrollo multiplataformas, pero solo unos pocos se han adoptado mundialmente. Algunos de los frameworks para el desarrollo multiplataforma más importantes actualmente son Xamarin, React Native, Flutter y Cordova.

1.4.1 Xamarin

Xamarin [4,5] es un framework para desarrollo híbrido creado en 2011 y comprada en 2016 por Microsoft. Se caracteriza por utilizar C\# como único lenguaje, a la vez que permite el acceso al API nativo.

Puntos fuertes:

Interoperabilidad con otros lenguajes, por lo que es posible utilizar librerías desarrolladas para aplicaciones nativas. El rendimiento de las aplicaciones construidas con este software es bastante alto, lo que permite utilizarlo incluso para juegos exigentes.

Puntos débiles:

Acceso limitado a algunas librerías importantes en su versión gratuita. El núcleo de su creación de interfaz de usuario no es móvil, por lo que los desarrolladores necesitan emplear tiempo en cada plataforma. El apoyo de la comunidad es menor que en otros frameworks. El único entorno de desarrollo en el que se puede utilizar es Visual Studio.

1.4.2 React Native

React Native [6] es un framework para el desarrollo híbrido creado en 2015 por Facebook. Utiliza JavaScript como único lenguaje de desarrollo y está estructurado en componentes. Cada componente tiene una función render que indica cómo debe visualizarse. Las vistas se escriben con JSX, que es una extensión de JavaScript para la creación de vistas usando sintaxis HTML. En React Native, los elementos HTML se sustituyen por componentes REACT. React Native se encarga de renderizar los componentes tanto en IOS como en Android utilizando una API Objective-c o Java respectivamente.

Puntos fuertes:

Utiliza lenguaje Javascript, uno de los lenguajes más populares [7] y utilizado en gran cantidad de frameworks. Cuenta con una gran comunidad de usuarios para resolver dudas durante el desarrollo.

Puntos débiles:

El rendimiento de las aplicaciones desarrolladas con React Native es peor que el de las aplicaciones nativas.

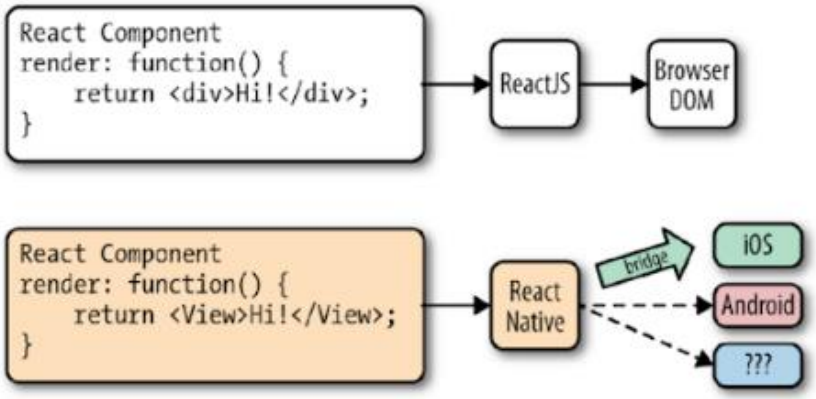

Figura 1: Renderizado en React Native

1.4.3 Flutter

Flutter [8,9] es un SDK open source creado en 2017 por Google para el desarrollo de aplicaciones Android e IOS a partir de un solo código base. Utiliza el lenguaje Dart y está basado en widgets. Los usuarios de flutter definen la lógica de la aplicación con Dart, la interfaz a partir de widgets y Flutter se encarga de renderizarlo con un motor 2D propio escrito en C++.

Puntos fuertes:

Ofrece un menor tiempo de desarrollo que otros frameworks similares. Las aplicaciones construidas con Flutter se ejecutan con código nativo compilado, por lo que el rendimiento es bastante bueno.

Puntos débiles:

Debido a su reciente creación, algunas funcionalidades están todavía en desarrollo. Además, por el momento no es compatible con plataformas de integración continua como Jenkins o Travis.

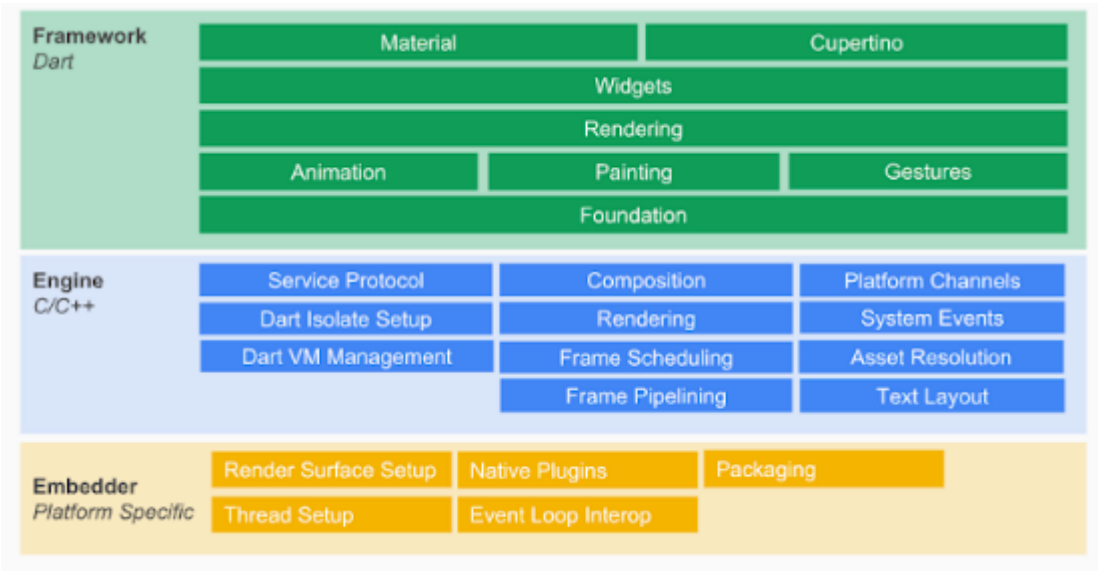

Figura 2: Estructura de Flutter

1.4.4 Apache Cordova

Apache Cordova [10] es un framework open source para el desarrollo de aplicaciones multiplataforma basado en webviews y en plugins. Las aplicaciones desarrolladas con apache Cordova se ejecutan dentro de webviews y son capaces de acceder a una gran variedad de funcionalidades del hardware mediante una API HTML5. Para aquellas funcionalidades a las que no se pueda acceder mediante HTML5, existe una serie de plugins que permiten su acceso desde JavaScript.

Puntos fuertes:

El lenguaje de desarrollo es el mismo que se utiliza para aplicaciones web, por lo que las empresas pueden tener un sólo equipo para desarrollar tanto aplicaciones móviles como aplicaciones web. Contiene una gran comunidad de usuarios para apoyar durante el desarrollo.

Puntos débiles:

El rendimiento de las aplicaciones que se ejecutan sobre webviews no es demasiado bueno. Además, los plugins suministrados por apache cordova no cubren todas las funcionalidades y es necesario utilizar plugins de terceros o crear tus propios plugins.

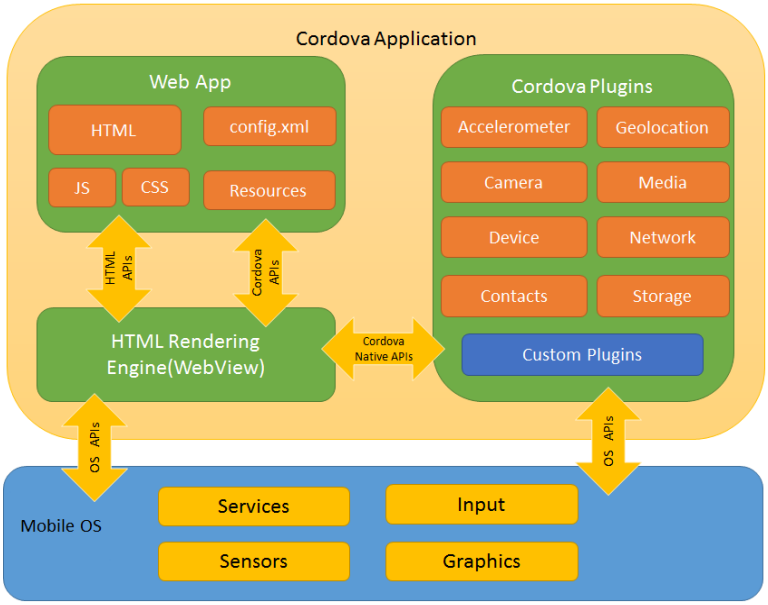

Figura 3: Estructura de Apache Cordova

1.5 Estructura del documento

Este documento se ha dividido en 7 apartados diferentes:

- **Introducción**: Explicación de por qué es necesario el proyecto, en qué consiste y análisis de los principales frameworks para el desarrollo de aplicaciones móviles multiplataforma.
- **Desarrollo de aplicaciones móviles multiplataforma**: Estudio de algunas de las principales alternativas para el desarrollo multiplataforma, en concreto, las progressive webapps y el desarrollo hibrido con webviews. Justificación de Ionic como framework para el desarrollo del proyecto y análisis de su estructura y funcionamiento.
- **Análisis y diseño**: Cuáles son los requisitos del proyecto, a qué público va dirigido, cuáles son los datos disponibles y descripción de las vistas que reflejan el diseño de la aplicación.
- **Implementación de la solución**: Este apartado describe la aplicación a un bajo nivel de abstracción. Algunos de los temas que trata son las herramientas utilizadas los problemas encontrados y sus soluciones.
- **Pruebas**: En este apartado se especifican cuáles han sido las pruebas realizadas para validar el correcto funcionamiento del sistema y el cumplimiento de las especificaciones.
- **Gestión del proyecto**: En este apartado se muestra la metodología empleada, el control de esfuerzos, un análisis de riesgos y la gestión de configuraciones de la aplicación.
- **Conclusiones**: en este apartado se analizan los resultados de los apartados anteriores para decidir si el trabajo ha sido satisfactorio y se realiza una valoración personal sobre el proyecto.

Además, se han incluido 3 anexos:

- **Mapas de navegación**: Creados durante la fase de diseño y que muestran de forma sencilla el funcionamiento de la aplicación sin entrar en detalles de estilo.
- **aGROSLab- Cuaderno de Explotación**: Es la versión web del proyecto y que cuenta con una funcionalidad parecida a la de la versión móvil incluso alguna más como la pantalla de estadísticas y la de detalle del recinto.
- **Evidencias de las pruebas**: Para demostrar que se han realizado las pruebas durante el desarrollo del proyecto se han incluido algunas imágenes de las herramientas y las pruebas utilizadas.

2 Desarrollo de aplicaciones móviles multiplataforma

Como ya se ha mencionado anteriormente, existen dos alternativas para desarrollar aplicaciones multiplataforma, aplicaciones híbridas y progressive webapps.

2.1 Progressive webapps

A efectos del usuario una aplicación móvil es aquella que cumple con los siguientes criterios [13]:

- **Estilo y sensación nativa**: El estilo varía en función de la plataforma en la que se ejecuta y los usuarios pueden navegar utilizando movimientos que les resulten naturales.
- **Velocidad y rendimiento**: Responde rápido al interactuar con el usuario.
- **Soporte offline**: La aplicación puede ser utilizada incluso cuando el usuario no dispone de cobertura de datos.
- **Instalable**: Los usuarios pueden lanzar la aplicación desde un icono en la pantalla de inicio.
- **Funcionalidad Nativa**: Funcionalidades típicas de una aplicación nativa, como pueden ser las notificaciones o el acceso al hardware del dispositivo, como la cámara o los sensores.

Tradicionalmente las aplicaciones web no eran capaces de cumplir estos requisitos, pero con los estándares web actuales sí que es posible. A las aplicaciones web que cumplen estos requisitos se les llama progressive webapps.

Las progressive webapps utilizan los llamados Service Workers [14, 15] para conseguir algunos de estos objetivos. Un Service Worker es básicamente un fichero JavaScript que se ejecuta de forma separada al hilo principal del navegador, interceptando las peticiones de red, almacenando o guardando datos en la cache o recibiendo notificaciones del servidor cuando la aplicación no está en uso. Los service workers se instalan la primera vez que el usuario accede a la dirección de la progressive web app. Un service worker puede ofrecer soporte offline interceptando las peticiones del usuario y devolviendo el contenido cacheado. El cacheado de contenido servirá también para mejorar el rendimiento de la aplicación.

Al tratarse de una tecnología emergente es importante comprobar el soporte en los navegadores más utilizados. Durante el 2018 muchos navegadores empezaron a dar soporte para progressive web apps [16].

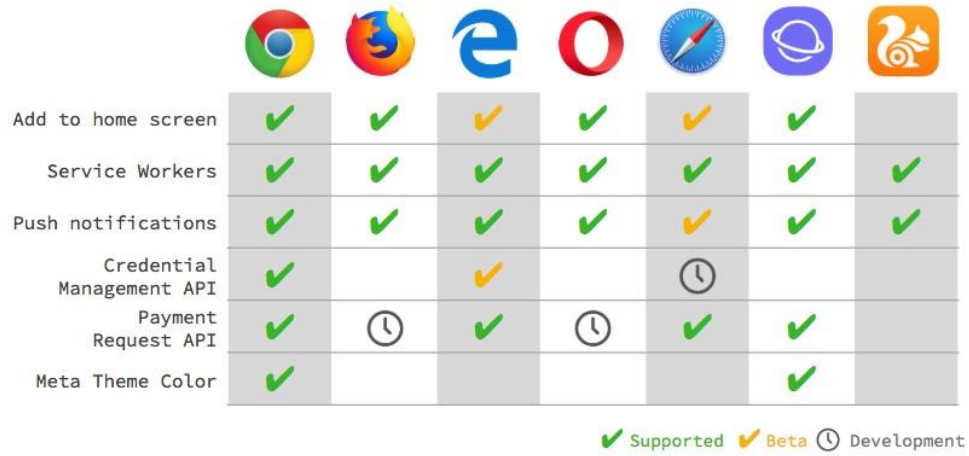

Figura 4: Soporte de los navegadores de PWA

Con las últimas versiones de los navegadores, es posible crear iconos para el uso de progressive web apps sin necesidad de subir la aplicación a Google Play o App Store. Sin embargo, sigue siendo deseable poder subir la aplicación a las tiendas oficiales para facilitar el descubrimiento de la aplicación y su monetización [17]. El proceso para subir la aplicación no debería ser un problema, ya que basta con un link de la progressive web app. Desde hace algunos años Microsoft permite subir progressive web apps a su tienda y recientemente Google play ha empezado a permitirlo.

Para que la aplicación pueda ser subida a Google play, esta debe cumplir con ciertos criterios, como obtener un rendimiento mínimo de 80/100 con la herramienta Lighthouse y que cumpla con las políticas de Google Play. Si la aplicación cumple con todos los criterios podrá ejecutarse como una TWA (trusted web activity) con Google Chrome a pantalla completa simulando una aplicación nativa.

Algunas de las ventajas que tienen las progressive web apps respecto al desarrollo híbrido son [18]:

- Pueden ser utilizadas desde navegadores o como aplicaciones de escritorio
- Ocupan menos espacio que las aplicaciones nativas
- No necesitan actualizaciones
- Pueden ser indexadas por motores de búsqueda.
- Despliegue rápido

2.2 Webview

Un Webview [20] es un navegador embebido dentro de una aplicación. Utilizar un Webview permite que las aplicaciones se desarrollen con tecnologías web y que además puedan empaquetarse como aplicaciones nativas y puestas en App Store o Google Play. Ambos, Android e IOS tienen un SDK que permite la renderización de aplicaciones Web mediante Webviews a la vez que mantienen el acceso nativo al resto del SDK.

Los Webviews modernos ofrecen una extensa API en HTML5 para acceder a la funcionalidad del hardware [21] como la cámara, los sensores o el bluetooth entre otras cosas. Aquellas funcionalidades que no están cubiertas dentro del Webview pueden ser utilizadas a través de una capa puente, normalmente utilizando plugins nativos que expongan un API en JavaScript. En las Aplicaciones Ionic , este puente es el plugin Ionic Webview.

2.3 IONIC

Ionic [11] es un framework Open Source para el desarrollo de aplicaciones móviles multiplataforma utilizando tecnologías web. La primera versión de Ionic se publicó en 2013 por Drifty Co y desde entonces ha ido creciendo con un gran apoyo por parte de la comunidad. Existen 4 versiones de Ionic y la última cuenta ya casi con 40.000 estrellas en Github [12] y es con la que se va a desarrollar el proyecto.

2.3.1 Comparación entre Ionic y otros frameworks multiplataforma

A la hora de elegir un framework para el desarrollo de aplicaciones móviles multiplataforma, las principales características a tener en cuenta son:

- **Rendimiento**: Flutter y Xamarín son los frameworks que mejor rendimiento ofrecen, ya que se compilan para convertirse en aplicaciones nativas. En Ionic las aplicaciones son hibridas, a pesar de ser aplicaciones nativas, el código de la aplicación se ejecuta dentro de un Webview, por lo que el rendimiento no es el mejor. En React Native el código es interpretado, por lo que el rendimiento tampoco es demasiado bueno.
- **Facilidad de uso**: Aunque es probable que alguno de los frameworks mencionados tenga una curva de aprendizaje mayor al resto, el principal factor a tener en cuenta es la experiencia previa con las tecnologías a utilizar. Ionic utiliza tecnologías web y la experiencia de desarrollo es similar a la que se esperaría en un desarrollo web estándar.
- **Diseño de GUIs**: Ionic es el frameworks que más completo en este sentido ya que ofrece una gran variedad de componentes con diseño nativo para Android e IOS. Otros como React Native también disponen de una variedad de componentes, pero el número de componentes es menor y algunos de estos son específicos de una plataforma.
- **Acceso al hardware nativo**: Poder acceder al hardware del dispositivo es fundamental en un framework de este tipo. En este aspecto, todos los frameworks mencionados ofrecen esa característica, algunos como React Native o Ionic mediante puentes que actúan de intermediarios y otros como Xamarín tienen acceso porque son aplicaciones nativas reales una vez han sido compiladas.
- **Comunidad y popularidad**: Ionic y React Native son los frameworks con mayor comunidad. Es posible que la comunidad de Flutter crezca en los próximos años ya que se trata de un framework bastante reciente.

Para el desarrollo de este proyecto se decidió utilizar Ionic que permite tanto desarrollo híbrido como progressive web apps. La razón para utilizar este framework en vez de los mencionados anteriormente es la experiencia del equipo de desarrollo de GeoSpatiumLab en desarrollo de servicios web. Además, el sistema a desarrollar no tiene requisitos estrictos en cuanto al rendimiento, por lo que su principal desventaja no supone un gran problema en este proyecto.

2.3.2 Trabajo con Ionic

2.3.2.1 IONIC progressive webapps

Desarrollar con este enfoque es el objetivo de Ionic framework y para ello el equipo de Ionic ha desarrollado Capacitor, una herramienta de Ionic para el desarrollo de progressive web apps y aplicaciones híbridas. La idea de Ionic es que capacitor sea el sucesor de apache cordova en un futuro. Sin embargo, aún falta un tiempo hasta que las progressive web app puedan sustituir completamente al desarrollo híbrido. Hasta el momento los estándares web no ofrecen tantas funcionalidades del hardware como el SDK nativo de los dispositivos. Además, el rendimiento de estas aplicaciones sigue siendo mejorable y sigue habiendo muchas restricciones para distribuir una progressive web app en las tiendas oficiales. Por estas razones Ionic permite también el desarrollo híbrido mediante Webviews.

Algunos ejemplos de progressive web apps desarrolladas con Ionic son Forbes, Tinder o la versión lite de Twitter.

2.3.2.2 IONIC Webviews

Ionic se utiliza normalmente para desarrollar apps que puedan ser descargadas tanto por el App store como por Google Play [19]. Para que la aplicación se ejecute sobre un Webview y pueda ser descargada en App Store o Google Play, Ionic utiliza apache Cordova. Si la intención del desarrollador es utilizar un enfoque híbrido, podría utilizar simplemente apache Cordova. Sin embargo, apache Cordova no ofrece herramientas para diseñar una UI con aspecto nativo [22], por lo que sigue siendo recomendable utilizarlo junto con Ionic .

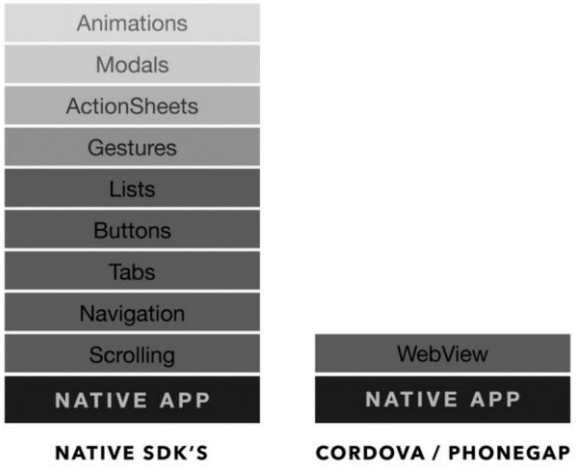

Figura 5: Diferencias entre Ionic y Apache Cordova

2.3.2.3 IONIC 4

La versión 4 de Ionic framework introduce algunas novedades respecto a su predecesor [23]:

- **Independencia de framework**: En versiones anteriores Ionic estaba fuertemente ligado a angular. En esta versión se ha rediseñado el framework para que trabaje como una librería Web autónoma, con soporte para los mayores frameworks JavaScript. El sistema se ha dividido en 4 módulos, un módulo core que permite su uso de forma independiente y otro para facilitar la integración con los frameworks angular, react y vue.
- **Estructura del proyecto**: La estructura del proyecto en Ionic se ha modificado para que coincida por la recomendada en el framework que se está utilizando.
- **Navegación**: En la versión anterior Ionic utilizaba su propio sistema de navegación. En la versión 4 si se utiliza Ionic junto con angular, se usa también su navegación con Angular Router.
- **Ciclo de vida de un componente** [24]: Al igual que con la navegación se ha modificado para que puedan utilizarse los eventos de Angular si se trabaja con ese framework.

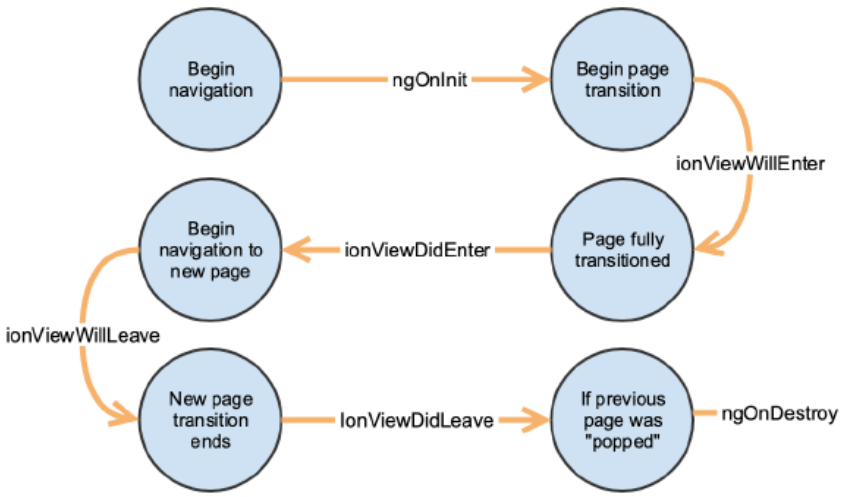

Figura 6: Ciclo de vida de los componentes Ionic

- **Capacitor**: En la versión anterior era necesario utilizar apache cordova para utilizar el SDK nativo e IOS o Android. En Ionic 4 es posible utilizar capacitor en vez de Cordova.

2.3.2.4 IONIC Components

Ionic dispone de una librería de componentes UI [25, 26], que son elementos reutilizables que sirven como bloques de construcción para una aplicación. Estos componentes siguen los estándares de la web y utilizan HTML, CSS y JavaScript.

Ionic utiliza diferentes estilos en función del dispositivo en el que se está ejecutando la aplicación. Los estilos siguen las pautas de cada plataforma. Cada plataforma tiene un estilo por defecto, pero Ionic está diseñado para que estos estilos puedan ser modificados sin dificultad.

Con la llegada de Ionic 4 es posible utilizar Ionic con otros frameworks además de Angular. Este cambio ha provocado algunos problemas de compatibilidad en los web components [27]. Para solucionar este problema, el equipo de Ionic ha desarrollado Stencil.

Stencil es un compilador para la generación de web components. Los desarrolladores definen el comportamiento y el estilo del componente web utilizando JSX (lenguaje utilizado también en React Native) y Stencil se encarga de generar el web component.

2.3.2.5 IONIC Native

Como ya se mencionó anteriormente, para acceder a las funcionalidades del hardware se utiliza un plugin llamado Ionic Webview. Este plugin proporciona acceso al SDK Nativo mediante plugins de apache cordova. Los plugins de cordova proporcionan una interfaz JavaScript mapeada al API del dispositivo. Apache cordova mantiene un conjunto de plugins para acceder a las principales funcionalidades del hardware del dispositivo. Además, existen una serie de plugins de terceros que completan el resto de la funcionalidad.

3 Análisis y Diseño

En este apartado se especifican los requisitos funcionales y los no funcionales, se muestran los prototipos de la interfaz de la aplicación utilizados en las primeras fases del proyecto, y se describen las vistas de la aplicación para comprender su arquitectura interna. En concreto, se ha incluido una descripción del modelo de datos utilizado, una vista de módulos, otra de componente y conector y otra de distribución.

Tal y como se ha indicado previamente, las aplicaciones móviles a desarrollar se engloban dentro de aGROSLab – Cuaderno de Explotación, línea de producto que surge, principalmente, para dar soporte al Registro de la Aplicación de Productos Fitosanitarios en las Explotaciones Agrícolas, de acuerdo a lo establecido en Real Decreto 1311/2012], de 14 de septiembre, por el que se establece el marco de actuación para conseguir un uso sostenible de los productos fitosanitarios. En el modelo operativo del sistema se plantean dos actores fundamentales:

- El asesor, técnico especialista en el diagnóstico de problemas de los cultivos, que saldrá al campo a supervisar las explotaciones y que, en base a sus observaciones, realizará prescripciones de los tratamientos que son necesarios aplicar.
- El agricultor, que recibirá las prescripciones del asesor para llevar a cabo, directamente o mediante persona/empresa contratada, los tratamientos que permitan proteger sus cultivos. Estos tratamientos, de acuerdo a la legislación mencionada, deben ser registrados para poder ser presentados ante las autoridades responsables, y deben ser tenidos en cuenta a la hora de realizar otros tratamientos o trabajos sobre los cultivos (por ejemplo, no se debería regar en un determinado periodo de tiempo posterior a la aplicación de un tratamiento para no perder efectividad, no debe cosechar hasta no pasado un determinado periodo de seguridad, etc).

Esta dualidad de actores se refleja en las explicaciones de las siguientes secciones.

3.1 Requisitos

3.1.1 Asesor

RF1	El sistema debe permitir al asesor descargarse los datos del servidor para que la aplicación pueda funcionar de forma offline.
RF2	El sistema debe permitir al asesor generar una prescripción.
RF3	El sistema debe permitir al asesor visualizar las prescripciones que ha creado.
RF4	El sistema debe permitir al asesor visualizar las parcelas de sus agricultores asociados en un mapa y en un listado.
RF5	El sistema debe permitir al asesor compartir prescripciones a través de mensajes y por correo electrónico.

RF6	El sistema debe permitir al asesor editar prescripciones que se han generado pero que todavía no se han subido al servidor.
RF7	El sistema debe permitir al asesor eliminar prescripciones que se han generado pero que todavía no se han subido al servidor.
RF8	El sistema debe permitir al asesor clonar prescripciones.
RF9	El sistema debe permitir al asesor visualizar las parcelas de cada una de sus prescripciones generadas.
RNF1	El sistema debe ser capaz de funcionar en un entorno sin cobertura de datos
RNF2	El mapa debe ser la parte central de la aplicación.
RNF3	La información asociada a la prescripción consta de ciertos datos elegidos en un listado y una foto opcional geolocalizada y con fecha.
RNF4	Una prescripción se realiza para un producto fitosanitario y una plaga. El sistema debe restringir que productos fitosanitarios se pueden utilizar en cada plaga y viceversa.
RNF5	En la creación de la prescripción el campo volumen solo será visible para ciertos productos fitosanitarios en función de su tipo de dosis (unidad)
RNF6	Los datos confidenciales como nombres o contraseñas deberán guardarse de forma segura.
RNF7	Para agilizar el proceso de descarga, el asesor debe poder elegir para cada productor con que cosechas quiere estar sincronizado.
RNF8	Al añadir producto a una prescripción se inicia la creación de una nueva prescripción con los mismos datos que la original, salvo por el producto fitosanitario, la plaga y la dosis utilizada.
RNF9	Si la dosis introducida está fuera del intervalo recomendado, se advertirá al asesor mostrando la dosis en rojo, pero no se impedirá la creación de la prescripción.
RNF10	El asesor debe ser capaz de ver todas las prescripciones que ha generado, incluso aquellas para las que no está sincronizado.

3.1.2 Agricultor

RF1	El sistema debe permitir al asesor descargarse los datos del servidor para que la aplicación pueda funcionar de forma offline.
RF2	El sistema debe permitir al agricultor generar tratamientos con o sin prescripción.
RF3	El sistema debe permitir al agricultor visualizar las prescripciones realizadas por su asesor en sus cultivos. Con toda la información asociada: foto, QR, ...

RF4	El sistema debe permitir al agricultor ver el histórico de los tratamientos realizados en sus cultivos
RF5	El sistema debe permitir al agricultor visualizar sus parcelas en un mapa.
RF6	El sistema debe permitir al asesor editar tratamiento que se han generado pero que todavía no se han subido al servidor.
RF7	El sistema debe permitir al asesor eliminar tratamientos que se han generado pero que todavía no se han subido al servidor.
RF8	El sistema debe permitir al asesor clonar tratamientos.
RF9	El sistema debe permitir al asesor visualizar las parcelas de cada una de sus prescripciones generadas.
RF10	Cuando un asesor genere una receta para un agricultor, el agricultor debe recibir una notificación. Al pulsar sobre la notificación el sistema debe iniciar una sincronización para que el agricultor tenga los datos actualizados.
RNF1	El sistema debe ser capaz de funcionar en entornos sin cobertura de datos.
RNF2	El mapa debe ser la parte central de la aplicación.
RNF4	Un tratamiento se realiza para un producto fitosanitario y una plaga. El sistema debe restringir que productos fitosanitarios se pueden utilizar en cada plaga y viceversa.
RNF5	En la creación del tratamiento, el campo volumen solo será visible para ciertos productos fitosanitarios en función de su tipo de dosis (unidad)
RNF6	Los datos confidenciales como nombres o contraseñas deberán guardarse de forma segura.
RNF7	Para agilizar el proceso de descarga, el agricultor debe poder elegir con que cosechas quiere estar sincronizado.
RNF8	Al añadir producto a un tratamiento, se inicia la creación de un nuevo tratamiento con los mismos datos que el original, salvo por el producto fitosanitario, la plaga y la dosis utilizada.
RNF9	Si un tratamiento se genera a partir de una prescripción, el producto fitosanitario y la plaga no pueden ser modificados.
RNF10	Si la dosis introducida está fuera del intervalo recomendado, se advertirá al agricultor mostrando la dosis en rojo, pero no se impedirá la creación del tratamiento.
RNF11	El agricultor debe ser capaz de ver todas las prescripciones y tratamientos que ha generado o recibido, incluso aquellas para las que no está sincronizado.

3.2 Interfaz

Al comienzo del proyecto se realizó un prototipo básico de las pantallas de la aplicación y de las transiciones entre estas pantallas a modo de mapa de navegación. El objetivo de este prototipo era poder enseñárselo a los clientes para comprobar que la idea que se tenía de la aplicación coincidía con la de ellos.

Una vez establecida la estructura de la interfaz de la aplicación, se realizaron prototipos más realistas de las pantallas más relevantes.

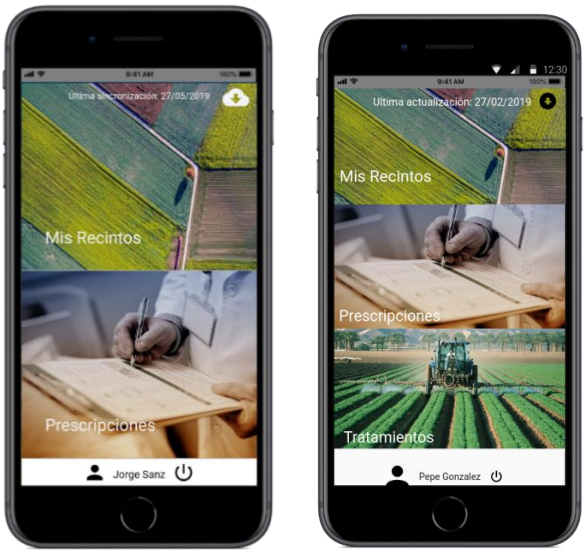

Figura 7: Prototipo pantalla de inicio

Para la pantalla inicial de la aplicación se decidió utilizar imágenes realistas para representar las principales funcionalidades de la aplicación. En la parte de arriba aparece la fecha de la última sincronización y en la parte de abajo el usuario logueado.

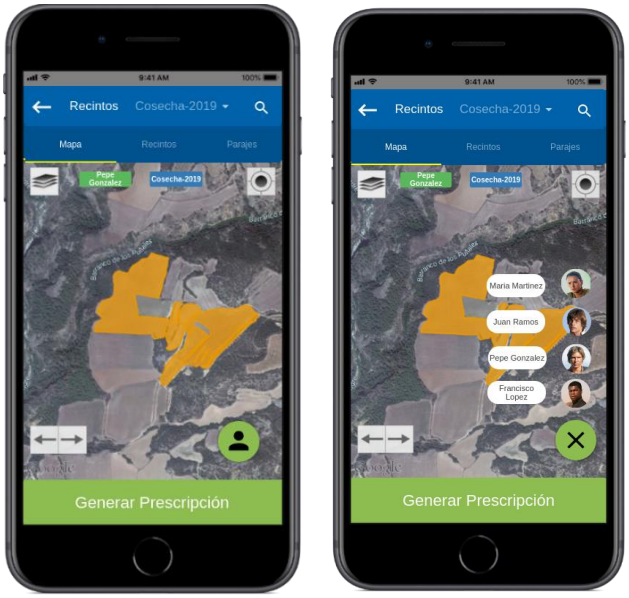

Figura 8: Prototipo pantalla “Mis recintos” (pocos productores)

En la pantalla de “Mis recintos” en el asesor, deben mostrarse únicamente los recintos del productor seleccionado. Por tanto, era necesario diseñar una interfaz para la selección del productor que resultase sencilla y agradable. Para ello se decidió utilizar un botón flotante desplegable. Durante el desarrollo se hicieron algunas mejoras sobre esta interfaz cambiando el color del botón a rojo y indicando el productor seleccionado con un label a la izquierda del botón flotante, de la misma forma que en el listado.

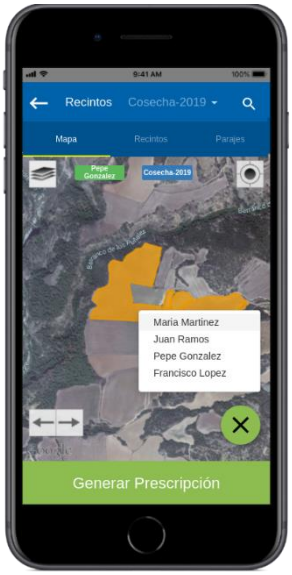

Figura 9: Prototipo pantalla “Mis recintos” (muchos productores)

Debido a que la interfaz anterior no resultaba usable si había muchos elementos, se diseñó también una interfaz alternativa para elegir entre 6 o más productores.

Figura 10: Pantalla de splash

Es importante que en algún lugar de la aplicación aparezca su nombre y logo para que el usuario pueda asociar la aplicación a una marca y distinguirla entre otras aplicaciones similares. Para no sobrecargar la pantalla de inicio se decidió utilizar la pantalla de carga con este propósito.

3.3 Modelo de datos

3.3.1 Agricultor

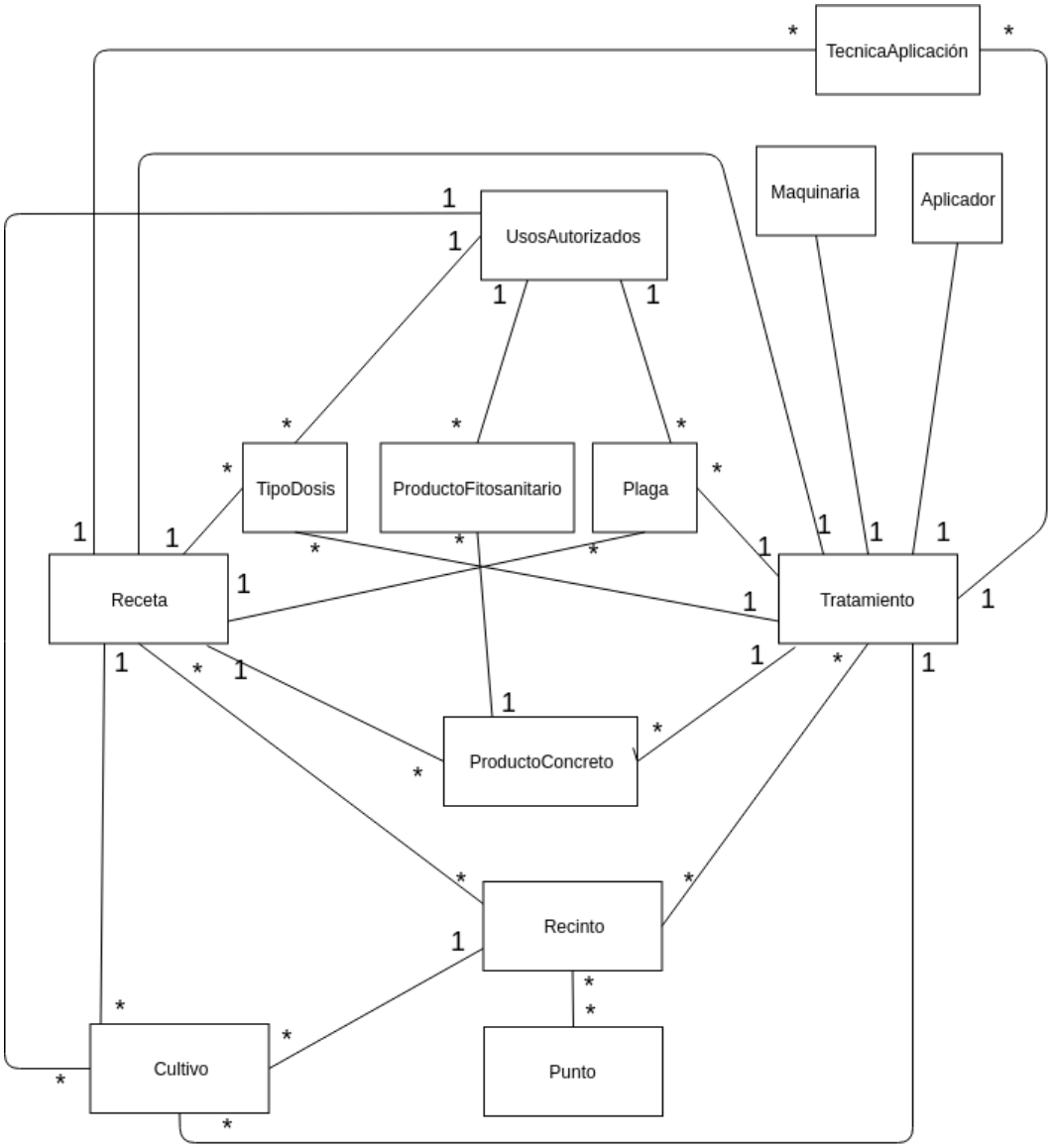

Figura 11: Modelo de datos agricultor

La generación y visualización de las recetas y de los tratamientos es la base de la aplicación del agricultor, por lo que su modelo de datos gira alrededor de estas entidades. Una receta o tratamiento está asociada a una plaga, a un producto fitosanitario (utilizado para combatir la plaga), a un cultivo y a un conjunto de recintos (Donde se aplica el producto). Una receta o tratamiento también se asocia a una técnica de aplicación, y en el caso del tratamiento, también a una máquina (utilizada para aplicar el producto) y a un aplicador (persona encargada de realizar el tratamiento). No todos los productos fitosanitarios pueden utilizarse sobre cualquier plaga o cultivo, la aplicación es responsable de gestionar qué productos se pueden utilizar en función de la plaga y del cultivo sobre el que se aplica. Los recintos están relacionados con su geometría, que se guarda en la tabla puntos.

3.3.2 Asesor

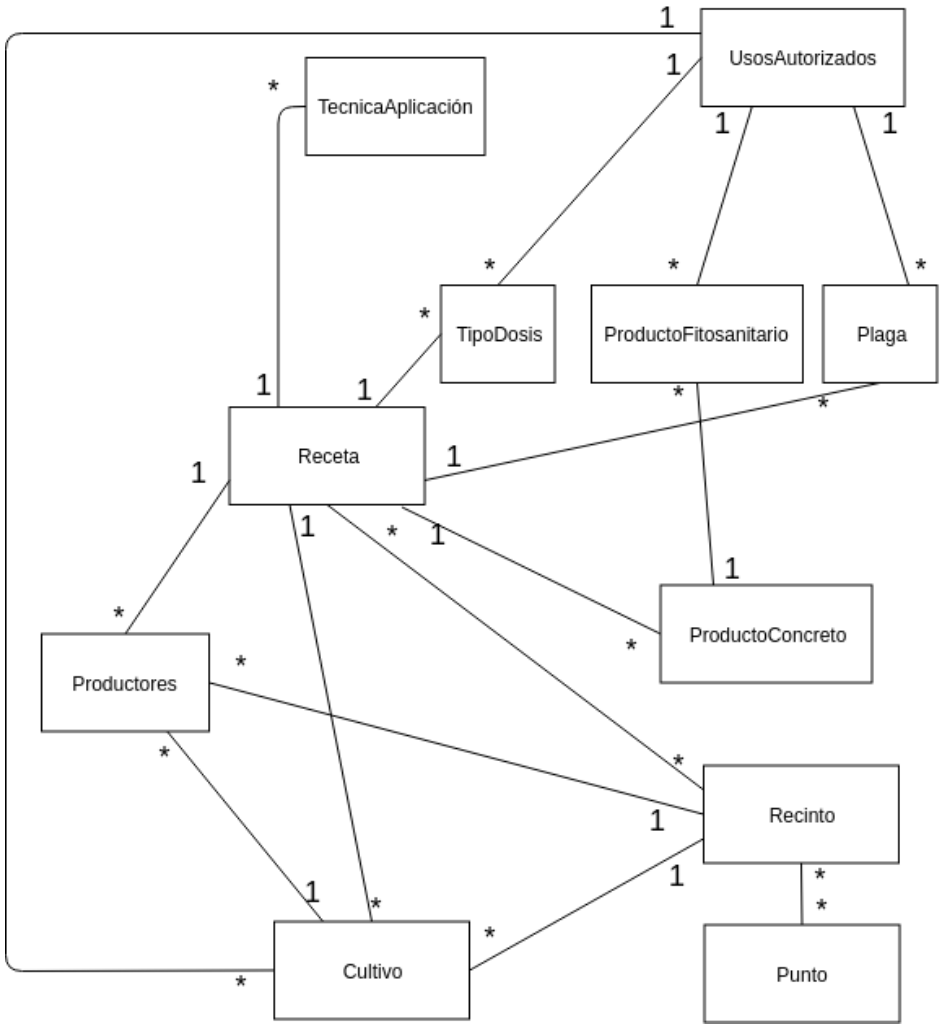

Figura 12: Modelo de datos asesor

El modelo de datos de la aplicación del asesor es parecido al del agricultor, pero con algunas diferencias. Un asesor no puede crear o visualizar tratamientos, por lo que en este modelo no existe la entidad Tratamiento. El asesor debe ser capaz de visualizar las parcelas y cultivos de sus productores y para eso existe la entidad Productores. Aunque este modelo pueda parecer más simple, el volumen de datos es mucho más grande.

3.3.3 Detalles

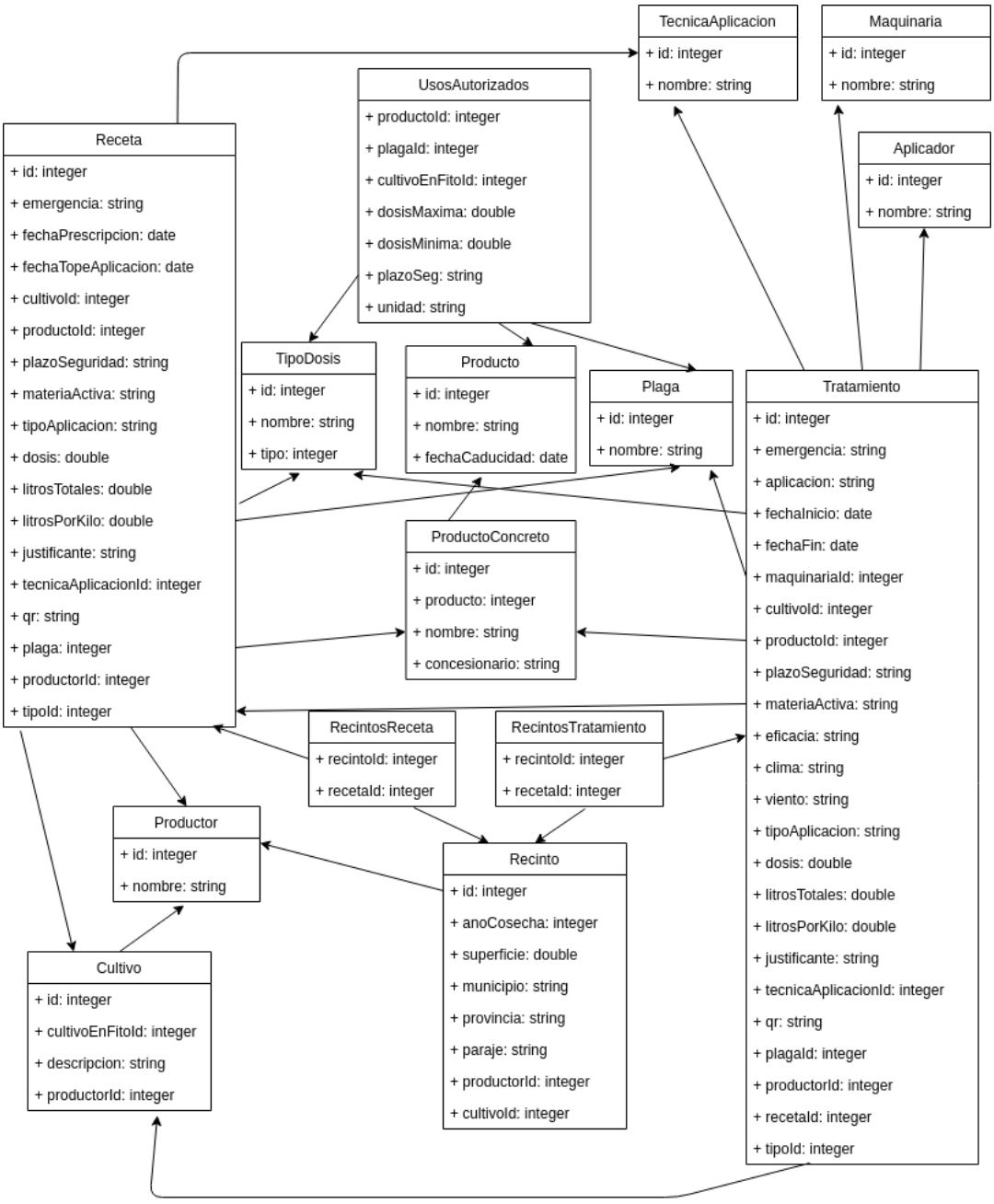

Figura 13: Modelo de datos detallado

Sobre los atributos de las entidades, conviene destacar:

- El atributo cultivoEnFitoId identifica el cultivo independientemente del productor.
- El atributo sigpac de la entidad Recinto identifica su geometría. Ese atributo está formado por la concatenación de los identificadores de provincia, municipio, zona, agregado y parcela.

3.4 Vista de Módulos

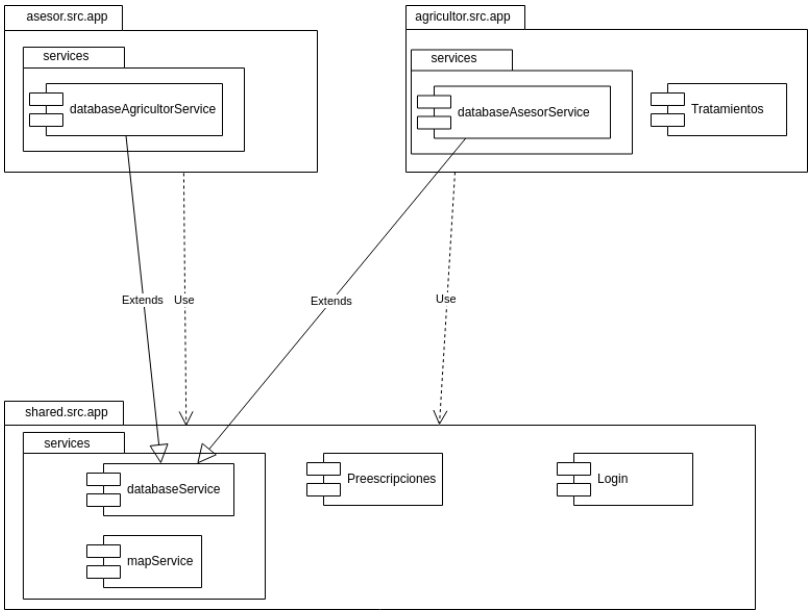

Figura 14: Diagrama de módulos

Para desarrollar las aplicaciones se ha seguido la estructura de la figura. Como ambas aplicaciones tienen funcionalidades y pantallas similares, tratarlos como dos aplicaciones independientes con código fuente separado hubiera sido un error ya que el coste de desarrollo y el de mantenimiento del proyecto hubiera sido mayor.

Para estas aplicaciones se han generado tres proyectos, dos para las aplicaciones y una para el código compartido. A pesar de que ambas aplicaciones son muy parecidas, es muy frecuente que tengan pequeñas diferencias en muchos componentes, para solucionar estas diferencias se ha utilizado la herencia. Existe un componente del que heredan para que cada aplicación pueda modificar el comportamiento o la vista del componente para que se ajuste a lo que necesita. Otros componentes no necesitan ningún cambio y pueden ser utilizados sin necesidad de herencia.

3.5 Vista CyC

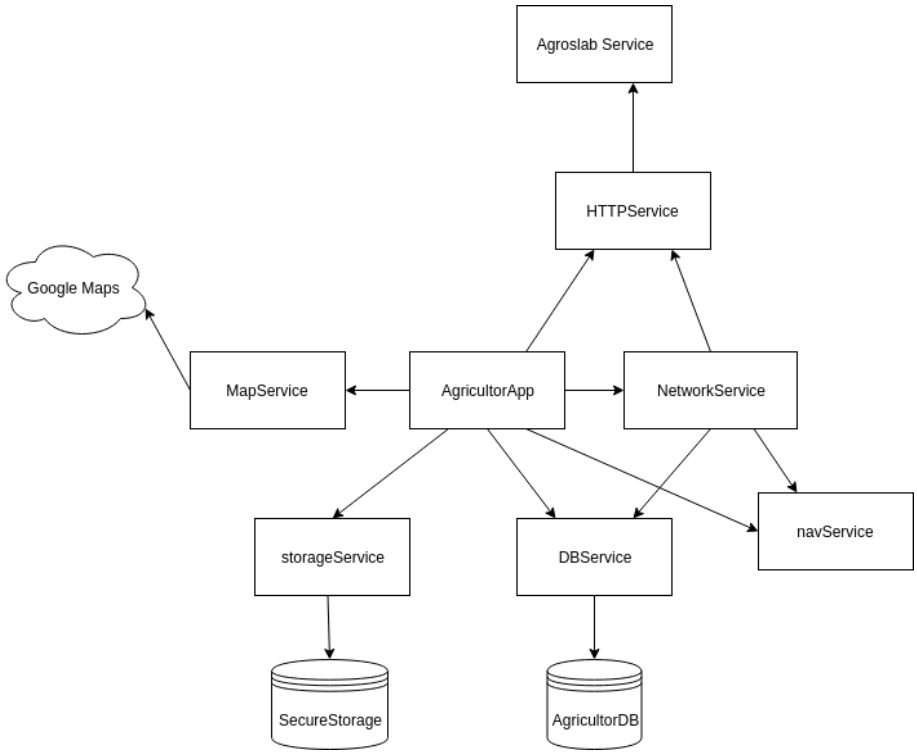

Figura 15: Diagrama de componentes y conectores

La vista de componentes y conectores en este proyecto es la misma para la aplicación del agricultor y para la del asesor y está formada por 7 componentes

- El componente AgricultorApp contiene las vistas y la lógica interna de la aplicación, pero necesita de otros servicios para interactuar con el exterior o para almacenar los datos.
- El componente HTTPService se encarga de realizar las peticiones al servidor de Agroslab para los datos fitosanitarios y al de GeoSpatiumLab para la información de parcelas.
- NetworkService detecta cambios en la red y si el usuario tenía tratamientos o prescripciones pendiente se encarga de subirlos.
- El componente navService sirve para compartir datos entre diferentes pantallas del sistema.
- DBService se encarga de almacenar los datos en una base de datos MySQL y de realizar consultas si otro componente lo necesita.
- El componente storageService sirve para guardar datos confidenciales o simplemente datos sin una estructura definida, como contraseñas, último login, estado del sistema, etc.
- El componente MapService se encarga de gestionar el mapa de google maps, de dibujar las parcelas y de seleccionarlas.

3.6 Vista Despliegue

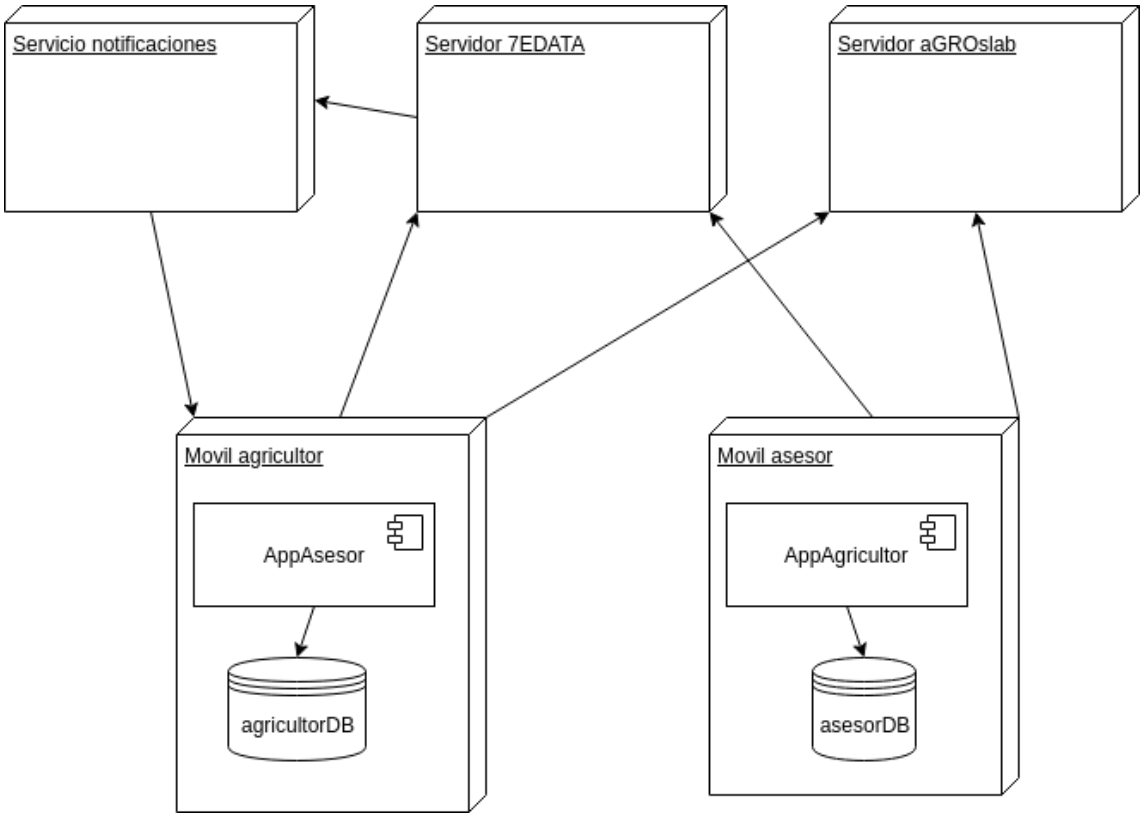

Figura 16: Diagrama de despliegue

Tanto la aplicación como la base de datos residen en el dispositivo de usuario. La comunicación con los servidores sólo es necesaria para la sincronización inicial y para la subida de los tratamientos.

Cuando un asesor genera una receta para un agricultor, el servidor de agroslab genera una notificación en el servicio de notificaciones (Firebase cloud messaging en Android y APN en IOS), que la transmite al dispositivo del agricultor.

4 Implementación

En este apartado se describen aspectos concretos de la implementación, como tecnologías utilizadas o problemas encontrados durante el desarrollo.

4.1 Mapas y Parcelas

Para la vista de "Mis recintos" se ha utilizado un plugin de Google Maps para Ionic 4 [28]. Este plugin permite entre otras cosas:

- Dibujar polígonos en un mapa y asociarlos a funciones que se ejecuten cuando el usuario pulse en el polígono.
- Mostrar la localización del usuario.
- Cachear las zonas visitadas por el usuario para que puedan ser accedidas más adelante sin necesidad de red.

Para no tener que depender de Google y además poder descargar los mapas para su uso offline, se consideraron otras opciones como Mapbox o openstreetmap.

Mapbox [29] permite la descarga de zonas de forma gratuita. Sin embargo, el tamaño de estas zonas era demasiado pequeño y además ocupaba mucho espacio. Está opción está más dirigida a aplicaciones que trabajen en un área más pequeña.

Openstreetmap [30] es un proyecto colaborativo para la creación de mapas libres y editables. En España existen algunos mapas que podrían haber sido adecuados para la aplicación. Sin embargo, o pesaban demasiado para un nivel de zoom aceptable o tenían muy pocos detalles y no aportaban nada.

4.2 Seguridad

A pesar de que las aplicaciones desarrolladas no contienen en general datos confidenciales, sí que es necesaria cierta seguridad para el almacenamiento de las credenciales. Hay que tener en cuenta que cada usuario solo almacena sus propios datos, por lo que el objetivo de esta seguridad es proteger sus datos frente a ataques externos.

Para guardar estos datos se ha utilizado un plugin diferente, llamado native-storage [31]. Este plugin guarda los datos en un espacio accesible solo por la propia aplicación. A pesar de ello, desde la propia página del plugin recomiendan utilizar algún mecanismo de cifrado para proteger aún más los datos

Los datos confidenciales almacenados se han encriptado con un algoritmo AES (Advanced Encryption Standard) [32]. AES utiliza una clave simétrica para la encriptación y desencriptación de los datos. La clave elegida se genera de forma dinámica a partir de la fecha del primer login del usuario.

4.3 Sincronización

Para que el agricultor o asesor pueda trabajar de forma offline, es necesario que antes descargue los datos del servicio. Esta operación está formada por múltiples peticiones y es bastante costosa. Tras estudiar las peticiones de la operación se ha encontrado que dos de ellas son las que consumen más tiempo por un amplio margen. Estas peticiones son:

- **Productos fitosanitarios**: Esta petición devuelve un listado de los productos fitosanitarios que el agricultor puede utilizar en sus cultivos junto con las plagas a las que se puede aplicar cada producto. La petición se realiza sobre un servidor externo (agroslab) sobre el que no tenemos control. El tiempo de respuesta de esta petición depende del número de cultivos.

- **Parcelas**: Información de la geometría asociada a los recintos. Cada geometría es un conjunto de puntos que componen la parcela. La petición se realiza sobre un servidor interno sobre el que sí que tenemos control. El tiempo de respuesta de esta petición depende del número de parcelas.

Para reducir el tiempo de descarga se han realizado las siguientes medidas:

- **Paralelización en la descarga de parcelas y productos fitosanitarios**: Como ambas peticiones se realizan sobre dos servidores diferentes, es posible, paralelizar estas peticiones sin saturar los servidores.

- **Peticiones de parcelas concurrentes**. Hay un máximo en el número de parcelas que el servidor puede devolver en una petición, por lo que las parcelas se piden en lotes de dicho tamaño. Si se envían peticiones concurrentes al servidor de parcelas, es posible reducir el tiempo de respuesta aprovechando los múltiples procesadores del servidor. Sin embargo, si se usan demasiadas peticiones concurrentes, el servidor podría saturarse, por lo que es necesario encontrar un término medio. Como el servidor de parcelas es interno, sabemos la carga que recibe con estas operaciones.

Para elegir el número de peticiones concurrentes se realizaron algunas pruebas. En cada sincronización se indican dos tiempos, el primero es el tiempo de descarga de las geometrías y el segundo el tiempo de descarga de los productos fitosanitarios.

	1783 parcelas y 315 cultivos (Todo)	900 parcelas y 143 cultivos (Todos los productores disponibles, una cosecha cada uno)	60 parcelas y 6 cultivos (Todas las cosechas del agricultor de prueba)
1 petición	3:30/3:00 s	2:00/1:40 s	24/20 s
2 peticiones concurrentes	2:10/2:10 s	1:40/1:40 s	20/20 s
4 peticiones concurrentes	2:10/2:10 s	1:40/1:40 s	20/20 s

Al utilizar dos peticiones concurrentes el cuello de botella pasa a ser la descarga de productos fitosanitarios, por lo que no tiene sentido seguir aumentando.

- **Cacheado de parcelas**: La geometría asociada a las parcelas se actualiza cada año, pero las parcelas ya existentes no se modifican, por lo que una vez que el agricultor las ha descargado, ya no es necesario que vuelva a hacerlo. Al reducir el número de peticiones de parcelas reducimos la carga del servidor.

- **Selección de los productores y cosechas a sincronizar**: Hasta ahora el asesor tenía que descargar la información necesaria para cada uno de los agricultores con los que trabaja, lo que suponía un tiempo de descarga alto. Sin embargo, es posible que el asesor no necesite estar sincronizado con todos ellos, sino solo con los que trabaja habitualmente. Del mismo modo, es posible que tampoco le interese la información de cada año de sus agricultores, sino solo la del año más reciente. Permitiendo elegir los productores y los años a descargar el tiempo de descarga se reduce sustancialmente.

4.4 Flavors

En Android un flavor [33] es una variante de configuración que permite generar una versión diferente de la aplicación sin tener que gestionar dos proyectos diferentes. Un caso bastante habitual es utilizar los flavors para compilar una versión gratuita con contenido limitado y otra de pago que incluya más contenido.

En este proyecto se han flavor para tener versiones personalizadas de la aplicación en función del cliente. En algunos casos es necesario cambiar el icono de la aplicación, la pantalla de splash, algún título o incluso alguna vista.

Actualmente Ionic no ofrece esta funcionalidad, por lo que para realizarlo se ha utilizado gulp.js. Gulp [34] es una herramienta javascript para la construcción de sistemas en desarrollo web. Con Gulp, es posible definir una serie de perfiles y para cada uno de ellos asociar tareas que se ejecuten en orden.

Para cada flavor existen unas tareas que sustituyen los recursos de la aplicación por los del flavor antes de la compilación.

4.5 Plugins utilizados

Los plugins [35] utilizado en la aplicación han sido:

- **Cordova-plugin-globalization:** Sirve para detectar el lenguaje predefinido en el dispositivo. En función del lenguaje se muestran los textos en inglés o en español-
- **Cordova-plugin-camera:** Permite tomar fotos desde la aplicación predefinida en el dispositivo. Utilizada para tomar fotos al crear una prescripción y guardarlas como base64, que es el formato requerido por el servidor de agroslab.
- **Cordova-plugin-geolocation:** Sirve para obtener la localización del usuario. Utilizado para asociar unas coordenadas al tomar la foto.
- **Ionic-native-google-maps:** Utilizado para la visualización de los mapas.
- **Cordova-plugin-advanced-http:**
- **Cordova-plugin-ionic-keyboard:**
- **Cordova-plugin-nativestorage:** Utilizado para guardar datos de forma segura
- **Cordova-plugin-network-information:** Sirve para obtener información de la red. Se utiliza para detectar cambios en la red y para evitar peticiones si el dispositivo no tiene red.
- **Phonegap-plugin-push:** Sirve para registrarse en los servidores de notificaciones de IOS y Android y para gestionar las notificaciones recibidas.
- **Cordova-sms-plugin:** Utilizado para la compartición de prescripciones por SMS en la app del asesor.
- **Cordova-plugin-email-composer:** Utilizado para la compartición de prescripciones por email en la app del asesor.
- **Cordova-sqlite-storage:** Utilizado para guardar datos de forma estructurada.
- **Cordova-plugin-statusbar:** Permite modificar el aspecto de la barra superior del dispositivo.

5 Pruebas

Dado que se trata de un sistema industrial, las pruebas son una parte fundamental del proceso. Es necesario asegurar que el sistema se comporta de acuerdo con lo especificado y que no existen comportamientos indeseados.

5.1 Pruebas de unidad

Las pruebas de unidad [36, 37] se utilizan para comprobar el correcto funcionamiento de una parte del código independientemente del resto del sistema. En los proyectos angular o Ionic el framework para las pruebas de unidad que se utiliza por defecto es Jasmine junto con Karma. Para este proyecto se decidió mantener ese framework para las pruebas de unidad.

Jasmine intenta describir los tests de forma que estos sean fácilmente legibles por humanos mediante etiquetas describe e it.

```
describe('Hello world', () => { (1)
  it('says hello', () => { (2)
    expect(helloWorld()) (3)
        .toEqual('Hello world!'); (4)
  });
});
```

Figura 17: Sintaxis pruebas con Jasmine

Para realizar pruebas sobre un componente de forma aislada, Jasmine permite la creación de Mocks que sustituyen a los componentes con los que interactúa. En programación orientada a objetos, un Mock es un objeto que imita el comportamiento de otro objeto. Jasmine ejecuta los tests sobre un navegador, donde muestra también los resultados.

Karma se utiliza para abrir el navegador y ejecutar los tests en él directamente desde la consola. Karma puede encargarse también de monitorizar el código y lanzar los tests siempre que se realicen cambios.

Ejecutar los test sobre un navegador externo puede resultar molesto, además si se utilizan sistemas de integración continua, es posible que no se ejecuten. Por estos motivos se ha decidido utilizar PhantomJS. PhantomJS [38] es un navegador sin interfaz gráfica usado para la automatización de la interacción con páginas web.

5.2 Pruebas de interfaz de usuario

Las pruebas de interfaz de usuario [39] consisten en comprobar que una aplicación cumple con sus especificaciones desde su misma interfaz gráfica. Al realizar las pruebas en un entorno parecido al que los usuarios utilizan la aplicación, los resultados son muy fiables. En este proyecto se han realizado pruebas de interfaz de usuario tanto manuales, como automáticas.

5.2.1 Manuales

Para las pruebas manuales se han definido una serie de casos de prueba. Cada caso de prueba contiene una descripción de lo que se quiere probar, las condiciones iniciales, las acciones a realizar y el resultado esperado. Los casos de prueba se ejecutan sobre una variedad de dispositivos. Es preferible que los dispositivos sean lo más diferente posible entre ellos.

Para este proyecto los casos de prueba se han anotado sobre un documento excel. En total, se han definido más de 200 casos de prueba y se han utilizado 5 dispositivos diferentes. A continuación, se puede ver una muestra de parte de dicho documento.

Descripción	**Precondición**	**Acciones a ejecutar**	**Resultados esperados**
Login usuario incorrecto	No hay datos en la base de datos	Loguear usuario no existente	Mensaje de usuario o contraseña incorrecto
Login usuario de otro tipo	No hay datos en la base de datos	Loguear usuario asesor	Redirigir a home. No puede hacer nada.
Login usuario correcto	No hay datos en la base de datos	Loguear usuario agricultor	Redirigir a home
Login sin conexión	No hay red	Loguear usuario agricultor	Mensaje error inesperado

5.2.2 Automáticas

Las pruebas automáticas se han realizado con Appium [40]. Appium es una herramienta open source para la automatización de pruebas multiplataforma.

Appium tiene su propia filosofía que encaja bastante con el concepto de Ionic .

1. No deberías tener que recompilar tu aplicación para poder automatizar las pruebas: Appium utiliza por debajo frameworks de automatización que ejecutan las pruebas directamente sobre la aplicación. Estos frameworks son principalmente XCUITests para IOS y UiAutomator2 para Android.
2. No deberías tener que elegir un framework o un lenguaje para realizar las pruebas. Appium encapsula los frameworks utilizados en una única API llamada WebDriver. WebDriver especifica un protocolo cliente-servidor en el que los clientes pueden realizar peticiones HTTP al servidor independientemente del lenguaje en el que estén escritos.
3. Un framework de automatización de pruebas no debe reinventar la rueda. WebDriver es el protocolo utilizado por Selenium para la automatización en navegadores web. Actualmente es el estándar de facto para este tipo de pruebas, por lo que Appium decidió extender el protocolo para incluir algunos métodos útiles para la automatización de aplicaciones móviles.
4. Un framework de automatización de pruebas debe ser software libre. Appium es software libre.

Además, Appium ofrece una interfaz de escritorio para grabar pruebas sobre dispositivos y exportarlas a los principales clientes del protocolo webdriver.

5.3 Pruebas sobre dispositivos físicos

Para asegurar que el sistema funciona correctamente en dispositivos reales se ha utilizado Perfecto Mobile. Perfecto Mobile [41] es una infraestructura de prueba de apps basada en la nube. Perfecto Mobile usa dispositivos reales para realizar las pruebas solicitadas. Perfecto permite utilizar más de 100 dispositivos diferentes (IOS o Android) para realizar pruebas. Sobre estos dispositivos es posible acceder de forma manual, instalar la aplicación a probar y controlarlos de forma remota desde la interfaz de Perfecto Mobile. Además, Perfecto Mobile ofrece una segunda interfaz para comprobar el resultado de las pruebas. En ella es posible descargar videos o snapshots de las pruebas realizadas y comprobar en tiempo real el estado de la prueba.

Perfecto Mobile no es la única infraestructura de pruebas sobre dispositivos reales, existen otras opciones como Firebase TestLab (Google) o SauceLabs. Sin embargo, se decidió utilizar perfecto mobile porque es compatible con pruebas appium.

La compatibilidad con pruebas Appium es un aspecto muy importante para este proyecto, ya que gracias a ello podemos desarrollar pruebas multiplataforma y ejecutarlas sobre cualquier dispositivo disponible en Perfecto Mobile. De esta forma, una vez desarrolladas las pruebas en Appium, podemos ejecutarlas sobre cualquier dispositivo disponible en Perfecto Mobile. Es más, si se utilizase junto con integración continua sería posible que cada nueva commit en el repositorio se probase de forma automática sobre un conjunto de dispositivos. Por desgracia esto último no fue posible por limitaciones con la licencia de uso.

5.4 Integración continua

La integración continua [42] es una práctica que consiste en subir el código con cierta frecuencia en una rama compartida en la que se automatiza la construcción y el testeo de la aplicación. Aunque se trate de un proyecto individual, sigue siendo recomendable utilizar esta práctica para automatizar el testeo de la aplicación. Por ello se ha utilizado la herramienta de integración continua de Gitlab.

Como ya se ha mencionado anteriormente, las pruebas con appium no han podido ser automatizadas en cada push sobre el repositorio de gitlab, sin embargo, sí que se ha automatizado la construcción de la aplicación y la ejecución de las pruebas de unidad.

6 Gestión del proyecto

En este apartado se describen aspectos sobre la gestión del proyecto durante su desarrollo, como el control de esfuerzos, el análisis de riesgos o la gestión de configuraciones.

6.1 Metodología empleada

Durante este proyecto, al igual que cualquier otro realizado en Geoslab, se han utilizado metodologías ágiles. En concreto, se ha seguido el marco de trabajo llamado Scrum.

Scrum [43] es una metodología ágil que se caracteriza por:

- El trabajo se realiza en iteraciones de duración predeterminada. Al comienzo de cada iteración el equipo planifica las tareas a realizar y al final de cada iteración el equipo revisa lo que se ha realizado. Las tareas deben ser estimadas y priorizadas para asegurar que pueden ser realizadas durante la iteración.
- El desarrollo se realiza de forma incremental, es decir, al final de cada iteración siempre hay algo nuevo que poner en producción.
- Existen diferentes roles como:
    1. Dueño del producto: Persona que decide lo que se debe desarrollar y el orden en el que se realiza
    2. Scrum master: Persona o personas encargadas de que se cumplan las prácticas de scrum.
    3. Equipo de desarrollo: Encargados de diseñar, construir y probar el producto.

Para este proyecto se han realizado iteraciones de aproximadamente dos semanas de duración. Los roles durante el proyecto han sido:

- Dueño del producto: Juan Lopez de Larrínzar e Ivan Salvador (Compañeros en GeoSpatiumLab)
- ScrumMaster: Todos
- Equipo de desarrollo: Jorge Sanz (Yo)

6.2 Esfuerzos

Para el seguimiento del proyecto se ha utilizado Redmine. Redmine es una herramienta software libre que permite entre otras cosas contabilizar las horas empleadas y mostrar diagramas para visualizar el tiempo empleado.

El proyecto ha tenido una duración de 702 horas repartidas en 10 iteraciones entre junio y noviembre. Dichas horas se han repartido entre:

- **Formación**: 112 horas. Estudio de Ionic y de las tecnologías del stack multiplataforma como Appium o PerfectoMobile.
- **Diseño**: 68 horas. Entre otras cosas, definir la estructura del sistema, el modelo de datos y realizar el prototipado de la aplicación.
- **Desarrollo**: 229 horas: Implementar las funcionalidades de las aplicaciones.
- **Pruebas**: 198 horas. Pruebas manuales, automáticas, con Appium o con PerfectoMobile
- **Gestión del proyecto**: 20 horas. Reuniones entre iteraciones Scrum para revisar lo realizado y definir lo que se va a realizar en la próxima iteración
- **Documentación**: 75 horas. Rellenar la memoria del trabajo de fin de máster.

6.3 Análisis de riesgos

Analizar daño y probabilidad de fallo		Probabilidad de fallo		
		Alto	Medio	Bajo
Daño en caso de fallo	Alto	A	A	B
	Medio	B	B	C
	Bajo	B	C	C

Para evitar problemas graves, se ha decidido dar una mayor clase de riesgo a los riesgos que en caso de materializarse produzcan un daño alto a pesar de que la probabilidad de que eso pase no sea muy alta.

Riesgo	Prob.	Daño	Clase de riesgo	Justificación	Estrategia
Retraso en el desarrollo del proyecto	Alto	Medio	B	Si se produce un retraso en el desarrollo del proyecto es posible que alguna de las funcionalidades no pueda realizarse y el proyecto no estaría completo	Realizar una buena planificación para que en caso de retrasarse con el desarrollo tener tiempo suficiente como para compensarlo.
Pérdida de alguno de los datos del proyecto	Medio	Medio	B	Si se pierde algún dato del proyecto se perderá tiempo en recuperarlo, ya sea descargándolo de nuevo o rehaciéndolo	Mantener tanto el código como la documentación en un sistema de control de versiones para que siempre sea posible recuperar una versión anterior del proyecto.
Actualización de Ionic a la versión 5	Medio	Alto	A	El equipo de Ionic sube versiones con cambios disruptivos cada 6 meses aproximadamente. Si la nueva versión es muy diferente, el proyecto perdería parte de su sentido.	Desarrollar el proyecto con Ionic 4 y añadir un apartado comentando los cambios de la nueva versión.
Desconocimiento de alguna las tecnologías empleadas	Alto	Bajo	B	El proyecto se está desarrollando utilizando herramientas del stack de Ionic con las que en su mayoría no he tenido la oportunidad de utilizarlas anteriormente	Parte del proyecto consiste en estudiar cómo funcionan estas herramientas. Este estudio se realiza previamente al desarrollo de la aplicación, por lo que para para entonces ya debería entender estas tecnologías.

6.4 Gestión de configuraciones

Para mantener un histórico de las distintas versiones de la aplicación y evitar la pérdida de datos se ha utilizado un sistema de control de versiones. El sistema de control de versiones utilizado en la empresa para proyectos anteriores era SVN. Sin embargo, la integración de SVN con algunas de las herramientas a utilizar resultaba problemática, especialmente en el caso de la integración continua. Además, las empresas actuales tienden cada vez más a usar GIT como sistema de control de versiones. Por tanto, se decidió utilizar GIT como sistema de control de versiones.

Al tratarse de código cerrado para una empresa, el alojamiento del código no suele ser gratuito. Tras evaluar algunas opciones se decidió utilizar gitlab.

Para almacenar la documentación del proyecto se utilizó una unidad de red compartida por la empresa. Sobre dicha unidad de red se realizan copias de seguridad de forma periódica.

6.5 Licencias

Al tratarse de un proyecto comercial es muy importante controlar las licencias de los recursos y de las herramientas utilizadas.

Para el desarrollo de las aplicaciones se ha utilizado Ionic que como ya se ha mencionado anteriormente es un framework libre.

Para el diseño de la aplicación se ha utilizado la herramienta Pencil que también es software libre. Las imágenes utilizadas en la aplicación provienen de Pixabay, una web para el intercambio de fotos con licencia Creative Commons.

Para la gestión de versiones y la integración continua se ha utilizado Gitlab que ofrece ambas funciones dentro de su plan gratuito.

7 Conclusiones

En este último apartado se realiza una retrospectiva del proyecto y se analiza cómo podría continuarse el proyecto.

7.1 Trabajo para un futuro

Una vez finalizado el proyecto sigue habiendo tareas a realizar, como pueden ser:

- **Labores de mantenimiento**: Solución de errores, mejoras estéticas, compatibilidad con nuevos dispositivos, etc.

- **Nuevas versiones**: En el futuro será necesario desarrollar nuevas funcionalidades sobre la aplicación actual. Estas funcionalidades se decidirán a partir de las necesidades que vayan surgiendo en los usuarios de la aplicación.

- **Desarrollo ecosistema aGROsLAB**: La aplicación desarrollada forma parte del ecosistema agroslab. Sin embargo, existen otras aplicaciones del mismo ecosistema que también necesitan una versión más actual y ser compatibles con sistemas IOS.

7.2 Valoración personal

Al realizar un trabajo de fin de máster una de las preocupaciones más comunes suele ser que ocurrirá con ese trabajo cuando esté terminado. Para mi ha supuesto un gran alivio saber que mi trabajo se utilizará cuando esté terminado y que no quedará en el olvido. Esa fue una de las principales motivaciones que tuve para empezar este proyecto y que me ha acompañado durante su desarrollo.

Por otro lado, el trabajo está dividido en dos partes, una de desarrollo del proyecto y otra de investigación, cada una de ellas con una forma de trabajo diferente. El poder variar el tipo de trabajo a realizar también ha ayudado a que la realización del proyecto sea más llevadera.

Realizar el proyecto en una empresa me ha ayudado a entender los altos criterios de calidad que tienen las aplicaciones destinadas a salir a un entorno de producción y la importancia de una buena gestión del proyecto.

En general considero que la realización del trabajo ha sido satisfactoria. Se han conseguido los objetivos propuestos al principio del proyecto y se han arreglado los problemas que han ido surgiendo durante el desarrollo.

Bibliografía

1. Boe.es. [online] Available at: https://www.boe.es/boe/dias/2012/09/15/pdfs/BOE-A-2012-11605.pdf
2. GeoSpatiumLab . *GeoSLab*. [online] Available at: https://www.geoslab.com/
3. Scholarspace.manoa.hawaii.edu. (2017). [online] Available at: https://scholarspace.manoa.hawaii.edu/bitstream/10125/41909/1/paper0760.pdf
4. Docs.microsoft.com. *What is Xamarin? - Xamarin*. [online] Available at: https://docs.microsoft.com/es-es/xamarin/cross-platform/get-started/introduction-to-mobile-development
5. Blog Brainhub.eu. *React Native vs Xamarin: Which Is Better? What Are Differences? - Blog Brainhub.eu*. [online] Available at: https://brainhub.eu/blog/react-native-vs-xamarin/
6. Google Books. (2019). *Learning React Native*. [online] Available at: https://books.google.es/books?id=274fCwAAQBAJ
7. Stack Overflow. *Stack Overflow Developer Survey 2019*. [online] Available at: https://insights.stackoverflow.com/survey/2019#most-popular-technologies
8. Flutter.dev. *Technical overview*. [online] Available at: https://flutter.dev/docs/resources/technical-overview
9. Es.wikipedia.org. (2019). *Flutter (software)*. [online] Available at: https://es.wikipedia.org/wiki/Flutter_(software)
10. Cordova.apache.org. *Architectural overview of Cordova platform - Apache Cordova*. [online] Available at: https://cordova.apache.org/docs/en/latest/guide/overview/index.html
11. En.wikipedia.org. *Ionic (mobile app framework)*. [online] Available at: https://en.wikipedia.org/wiki/Ionic_(mobile_app_framework)
12. GitHub. *ionic-team/ionic*. [online] Available at: https://github.com/ionic-team/ionic
13. Lucas, E. *Native or PWA? How to Choose the Right Approach for Mobile App Development*. [online] The Ionic Blog. Available at: https://blog.ionicframework.com/native-or-pwa-how-to-choose-the-right-approach-for-mobile-app-development/
14. Scitepress.org. [online] Available at: https://www.scitepress.org/Papers/2017/63537/63537.pdf
15. Google Developers. *Introduction to Service Worker | Web | Google Developers*. [online] Available at: https://developers.google.com/web/ilt/pwa/introduction-to-service-worker
16. Santoni, M. *iOS abre sus puertas a las Progressive Web Apps | GoodBarber*. [online] GoodBarber. Available at: https://es.goodbarber.com/blog/ios-abre-sus-puertas-a-las-progressive-web-apps-a607/
17. Brown, R. *Hot Take: Progressive Web Apps in the Google Play Store*. [online] The Ionic Blog. Available at: https://blog.ionicframework.com/hot-take-progressive-web-apps-in-the-google-play-store/
18. Ionic Framework. *Ionic Article: What is a Progressive Web App and Why You Need One?*. [online] Available at: https://ionicframework.com/resources/articles/what-is-a-progressive-web-app-and-why-you-need-one
19. Ionic Docs. (2019). *Core Concepts - Ionic Documentation*. [online] Available at: https://ionicframework.com/docs/intro/concepts#native-access
20. Pcmag.com. (2019). *WebView Definition from PC Magazine Encyclopedia*. [online] Available at: https://www.pcmag.com/encyclopedia/term/70186/webview
21. Ionic Docs. (2019). *Web View - Ionic Documentation*. [online] Available at: https://ionicframework.com/docs/building/webview
22. Netkow, M. *PhoneGap Devs: It's Time to Embrace a UI Framework*. [online] The Ionic Blog. Available at: https://blog.ionicframework.com/phonegap-devs-its-time-to-embrace-a-ui-framework/
23. Ionic Docs. (2019). *Migration Guide - Ionic Documentation*. [online] Available at: https://ionicframework.com/docs/building/migration
24. Ionic Docs. (2019). *Ionic Page Life Cycle - Ionic Documentation*. [online] Available at: https://ionicframework.com/docs/angular/lifecycle

25. Ionic Docs. (2019). *Core Concepts - Ionic Documentation*. [online] Available at: https://ionicframework.com/docs/intro/concepts
26. Ionic Framework. *Ionic Article: What is a UI Component Library?*. [online] Available at: https://ionicframework.com/resources/articles/what-is-a-ui-component-library
27. Stenciljs.com. *Stencil - A Compiler for Web Components - Stencil*. [online] Available at: https://stenciljs.com/docs/introduction
28. GitHub. (2019). *mapsplugin/cordova-plugin-googlemaps*. [online] Available at: https://github.com/mapsplugin/cordova-plugin-googlemaps
29. Mapbox. *Offline maps*. [online] Available at: https://docs.mapbox.com/help/troubleshooting/mobile-offline/
30. Wiki.openstreetmap.org. *ES:Descargar datos - OpenStreetMap Wiki*. [online] Available at: https://wiki.openstreetmap.org/wiki/ES:Descargar_datos
31. GitHub. *TheCocoaProject/cordova-plugin-nativestorage*. [online] Available at: https://github.com/TheCocoaProject/cordova-plugin-nativestorage#security
32. Es.wikipedia.org. *Advanced Encryption Standard*. [online] Available at: https://es.wikipedia.org/wiki/Advanced_Encryption_Standard
33. Medium. (2019). *Buenas prácticas con Product Flavors en Android - Parte 1*. [online] Available at: https://blog.kirei.io/buenas-practicas-con-product-flavors-en-android-parte-1-b3618ea1f386
34. En.wikipedia.org. (2019). *Gulp.js*. [online] Available at: https://en.wikipedia.org/wiki/Gulp.js
35. Ionic Docs. (2019). *Community Plugins - Ionic Documentation*. [online] Available at: https://ionicframework.com/docs/native/overview
36. Es.wikipedia.org. *Prueba unitaria*. [online] Available at: https://es.wikipedia.org/wiki/Prueba_unitaria
37. Ionic Docs. *Testing - Ionic Documentation*. [online] Available at: https://ionicframework.com/docs/building/testing
38. Phantomjs.org. *PhantomJS - Scriptable Headless Browser*. [online] Available at: https://phantomjs.org/
39. En.wikipedia.org. (2019). *Graphical user interface testing*. [online] Available at: https://en.wikipedia.org/wiki/Graphical_user_interface_testing
40. Appium.io. *Introduction - Appium*. [online] Available at: http://appium.io/docs/en/about-appium/intro/
41. Perfecto.io. *Web & Mobile App Testing | Continuous Testing | Perfecto*. [online] Available at: https://www.perfecto.io/
42. Es.wikipedia.org. *Integración continua*. [online] Available at: https://es.wikipedia.org/wiki/Integraci%C3%B3n_continua
43. Es.wikipedia.org. *Scrum (desarrollo de software)*. [online] Available at: https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software)

Anexo

A. Mapas de navegación

Durante el diseño de la aplicación se elaboraron mapas de navegación para identificar las pantallas de la aplicación y las transiciones entre ellas.

Agricultor

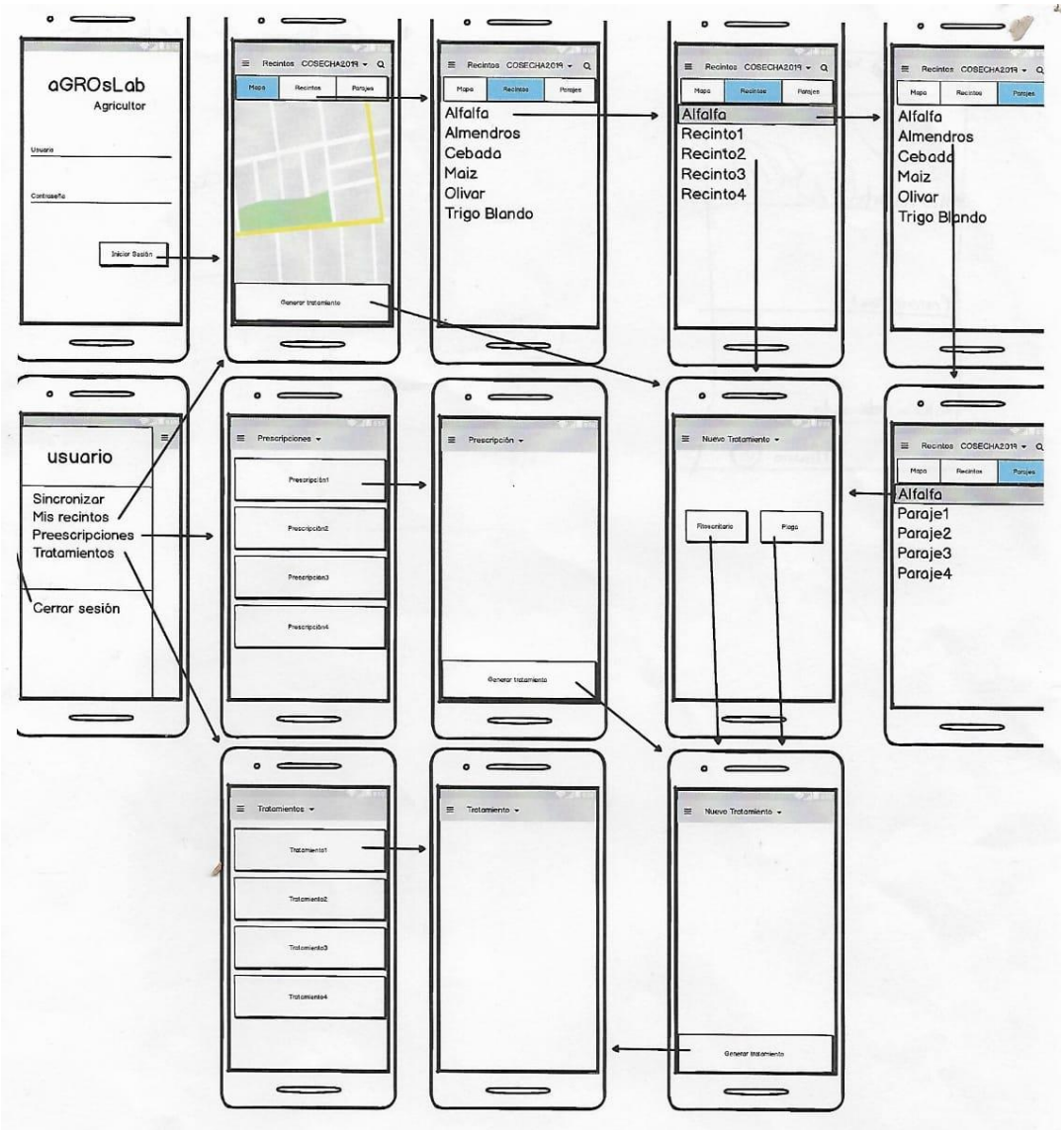

Figura 18: Mapa de navegación completo agricultor

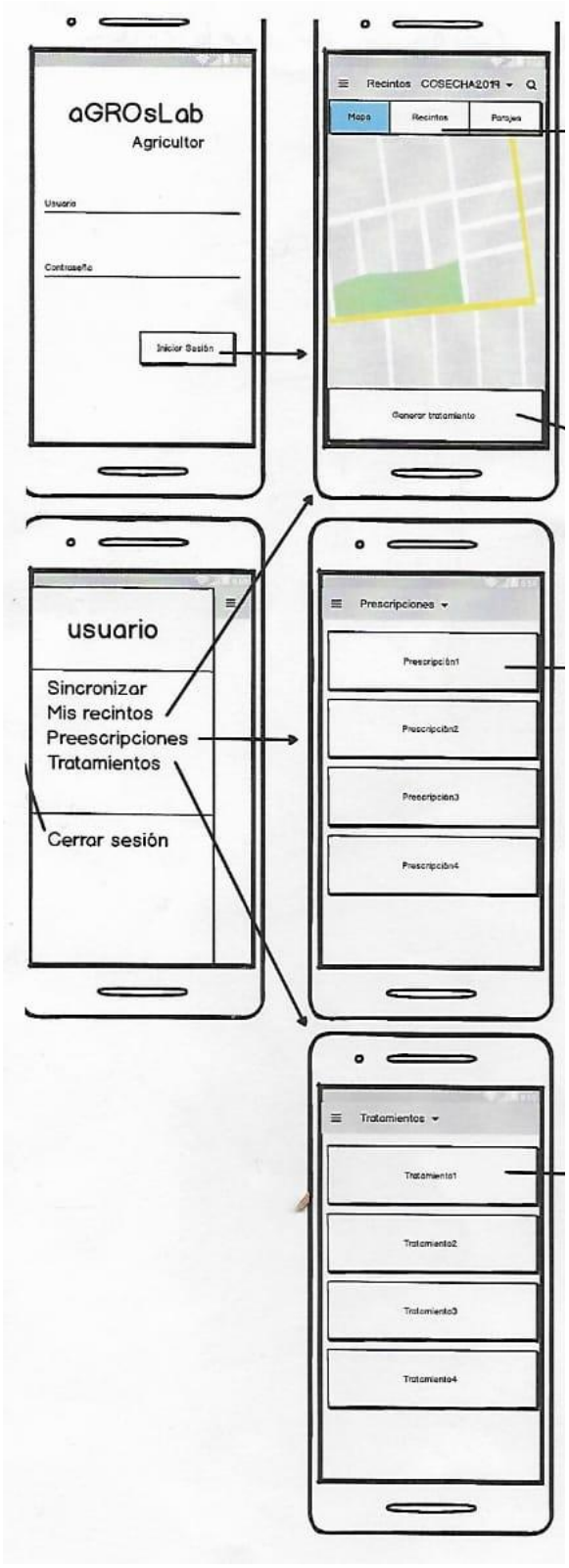

Figura 19: Mapa de navegación agricultor (1/3)

La idea inicial era que al iniciar sesión el agricultor fuera a la pantalla de mis recintos y desde un menú pudiese ir al listado de recintos y al de tratamientos. Finalmente se decidió redirigir a una pantalla inicial con las tres opciones.

Figura 20: Mapa de navegación agricultor (2/3)

Desde la pantalla de mis recintos el agricultor puede visualizar el mapa, el listado de recintos y el de parajes y generar un tratamiento de en uno o varios recintos. Al generar un tratamiento hay que decidir si empezar el tratamiento por fitosanitario o por plaga.

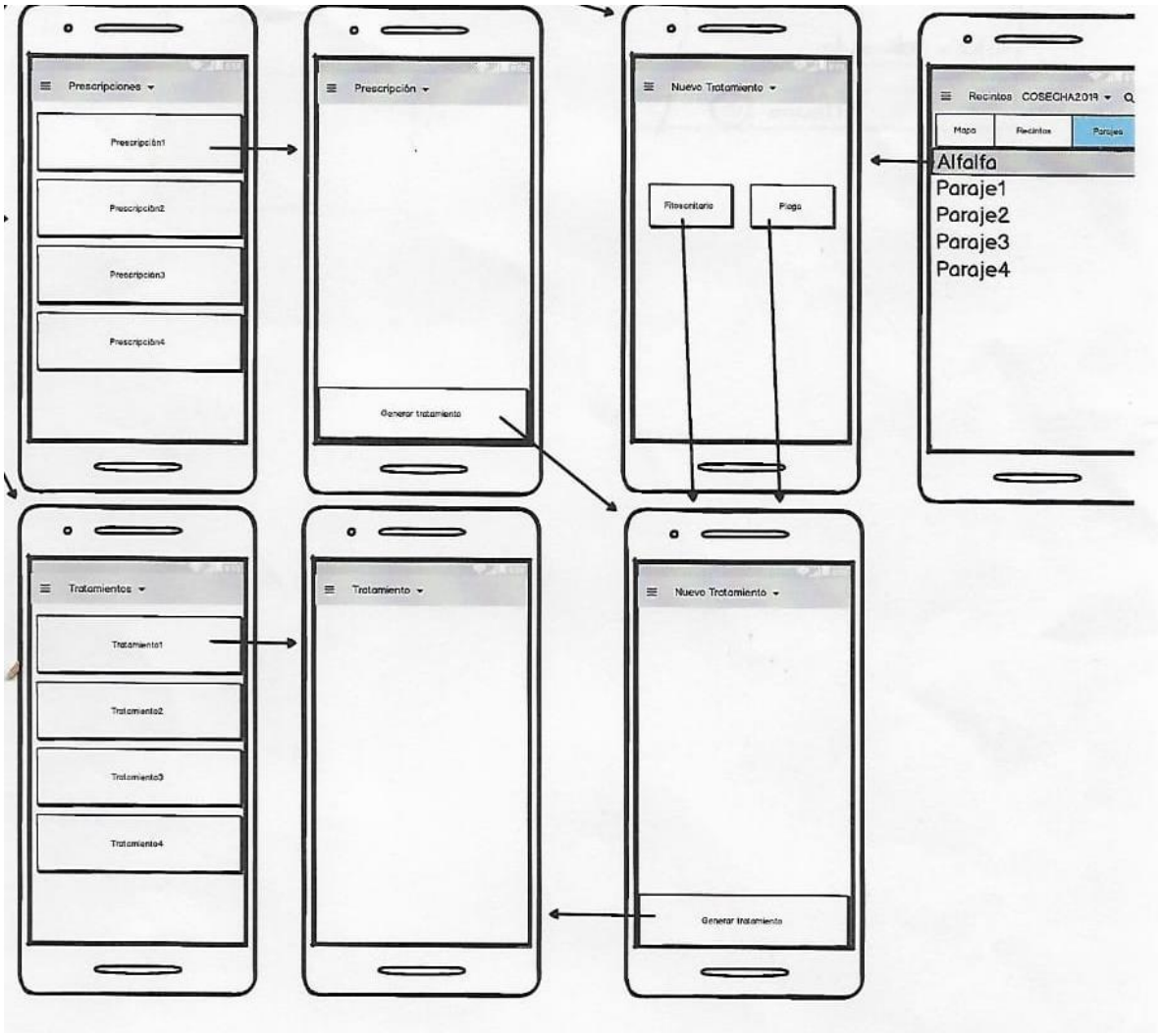

Figura 21: Mapa de navegación agricultor (3/3)

También es posible iniciar el tratamiento desde una prescripción. Una vez generado la aplicación redirige al usuario a la vista en detalle del tratamiento creado.

Asesor

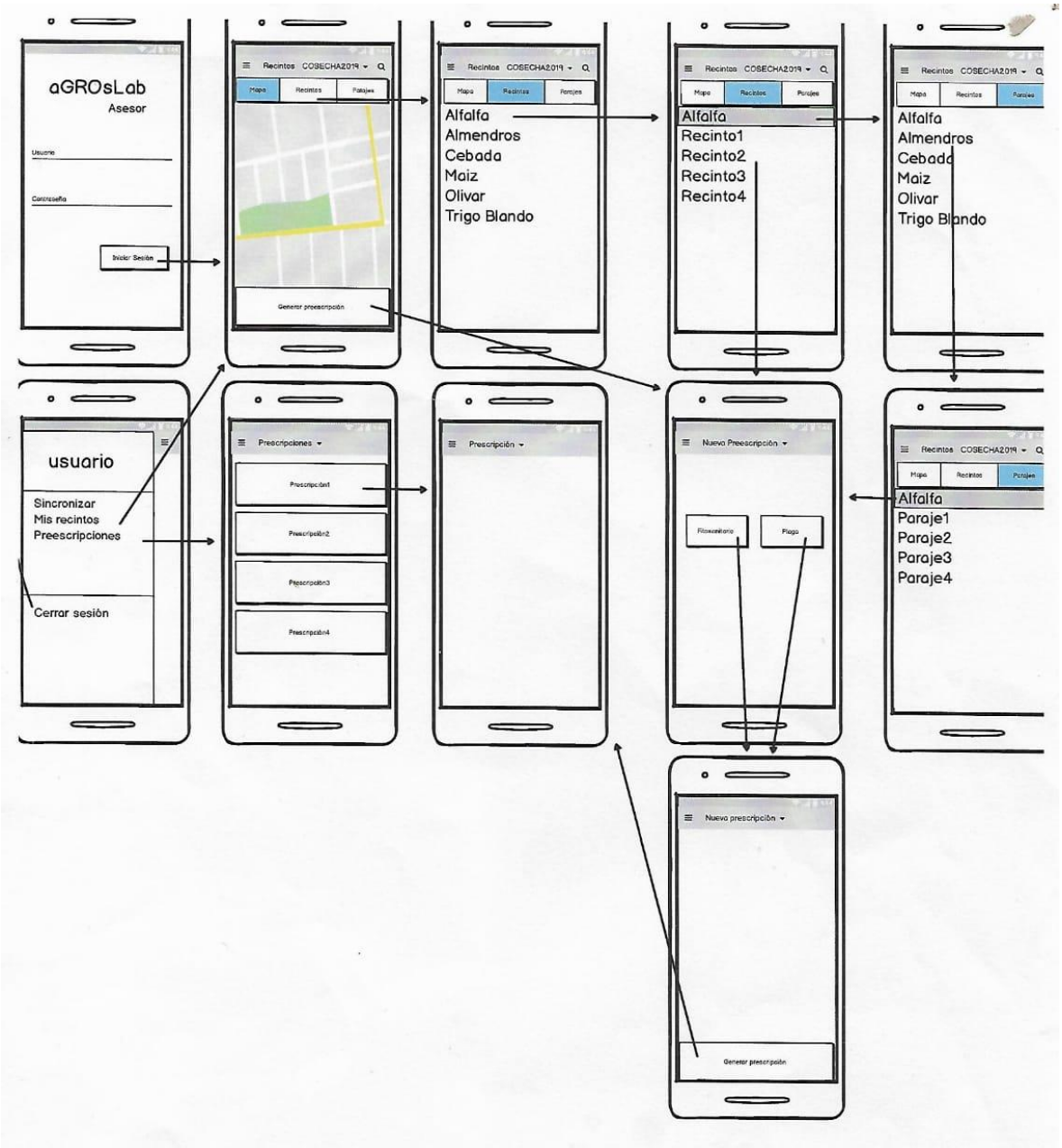

Figura 22: Mapa de navegación asesor

En la aplicación del asesor la estructura es bastante similar, solo que hay únicamente dos opciones, la vista de los recintos y el listado de prescripciones.

B. aGROSLab – Cuaderno de Explotación

aGROSLab – cuaderno de explotación es la versión web del proyecto y que cuenta con una funcionalidad parecida a la de la versión móvil incluso alguna más como la pantalla de estadísticas y la de detalle del recinto.

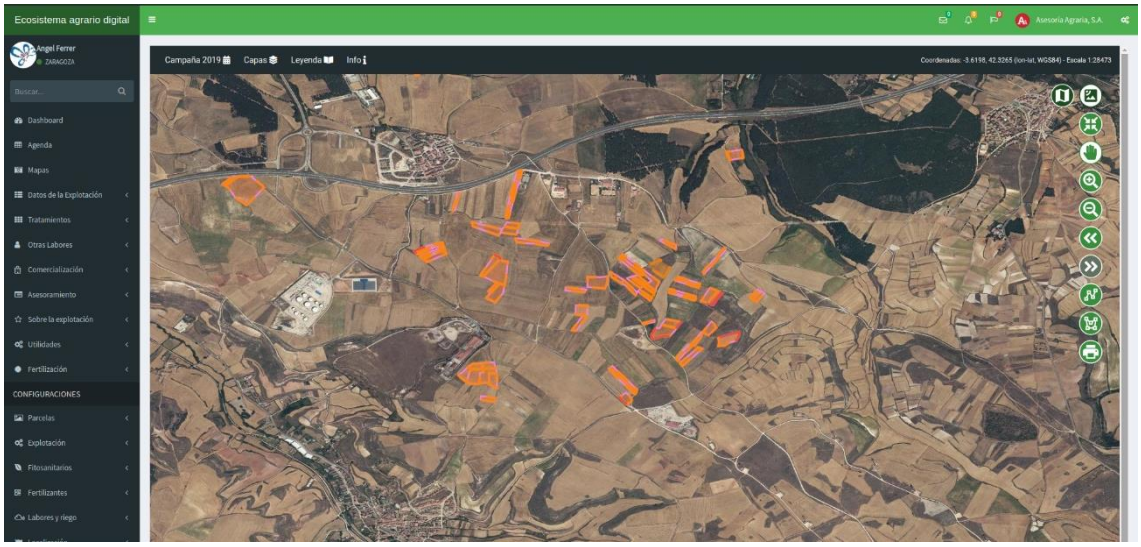

Figura 23: Visor GIS - Cuaderno de Explotación

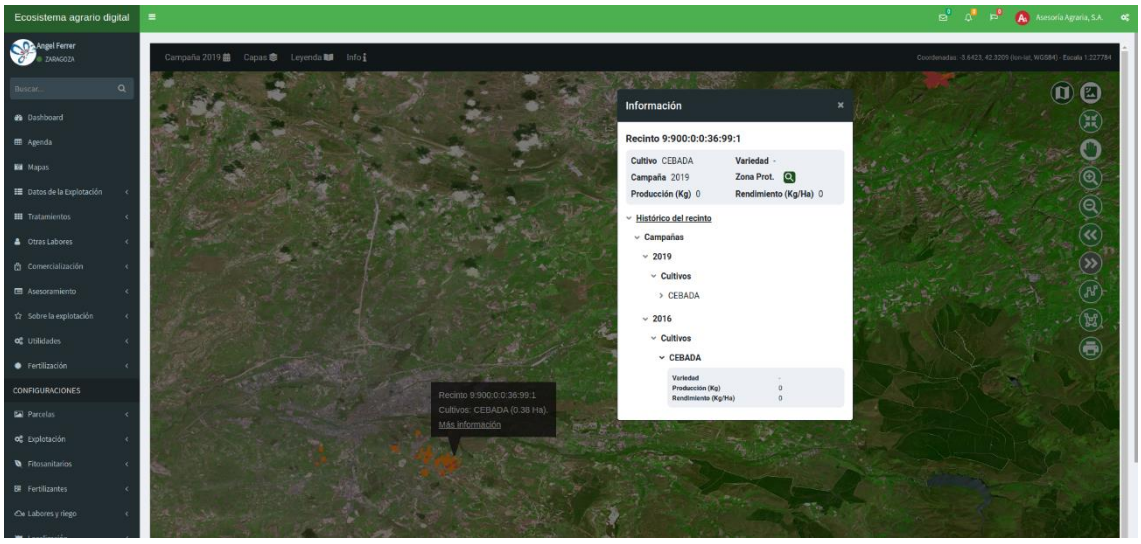

Figura 24: Detalle del recinto - Cuaderno de Explotación

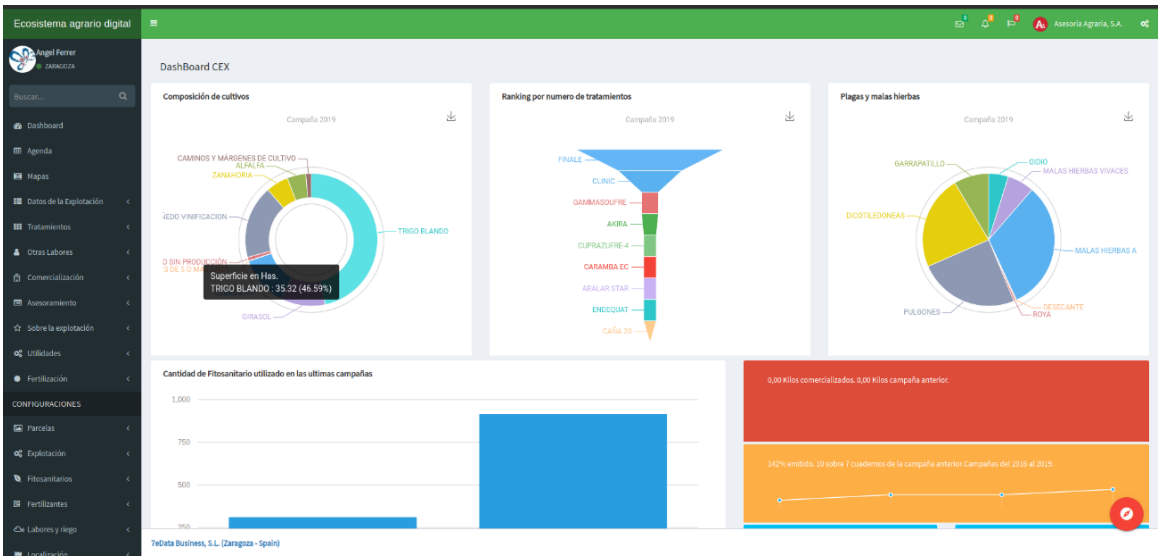

Figura 25: Vista de estadísticas - Cuaderno de Explotación

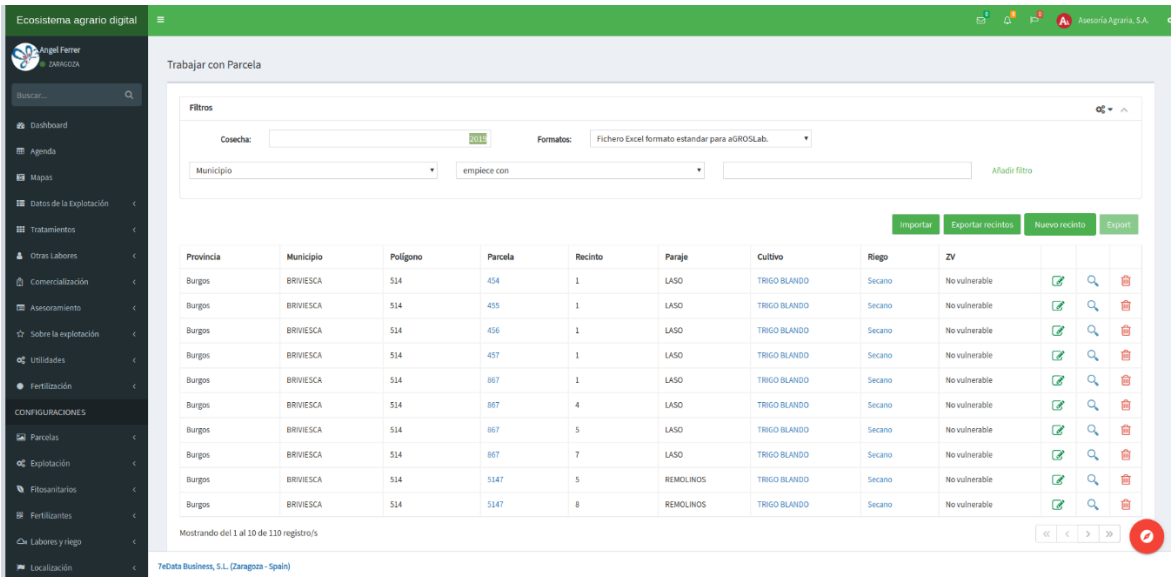

Figura 26: Vista de recintos - Cuaderno de Explotación

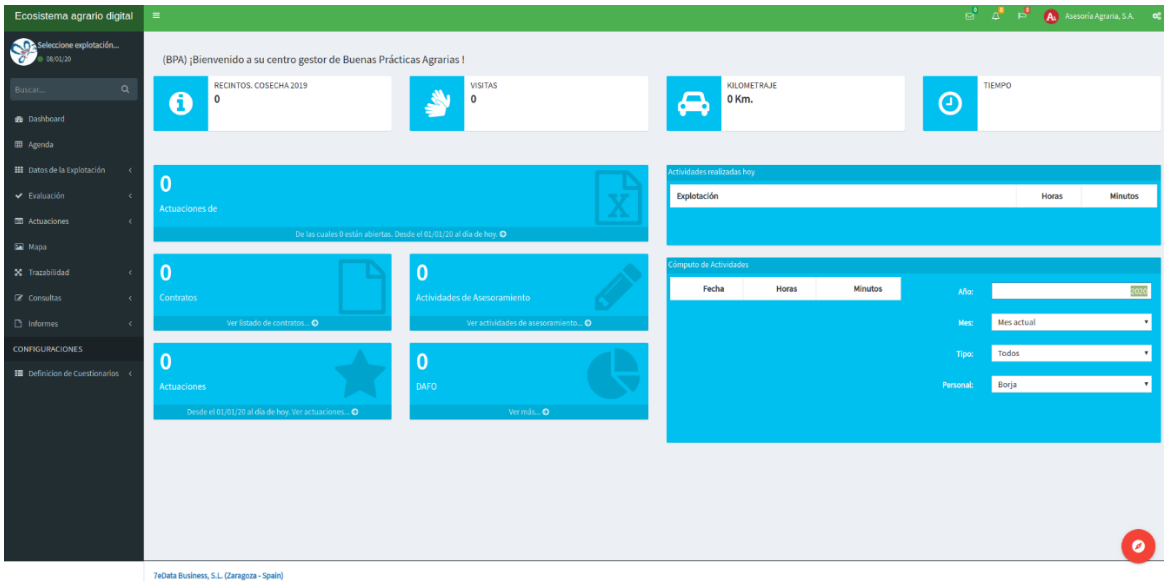

Figura 27: Vista de resumen - Cuaderno de Explotación

C. Evidencias de las pruebas

Para demostrar que se han realizado las pruebas durante el desarrollo del proyecto se han incluido algunas imágenes de las herramientas y las pruebas utilizadas.

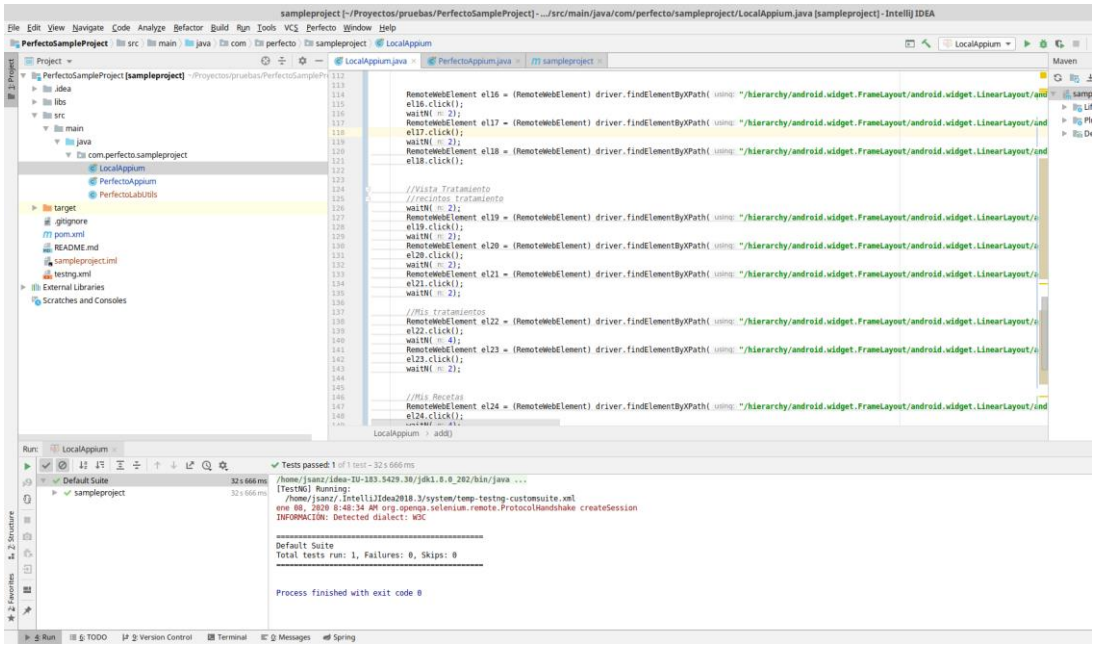

Figura 28: Proyecto Appium para realizar pruebas automáticas de forma local.

En la figura superior se muestra una imagen del proyecto Appium para la ejecución de las pruebas automáticas en un dispositivo local.

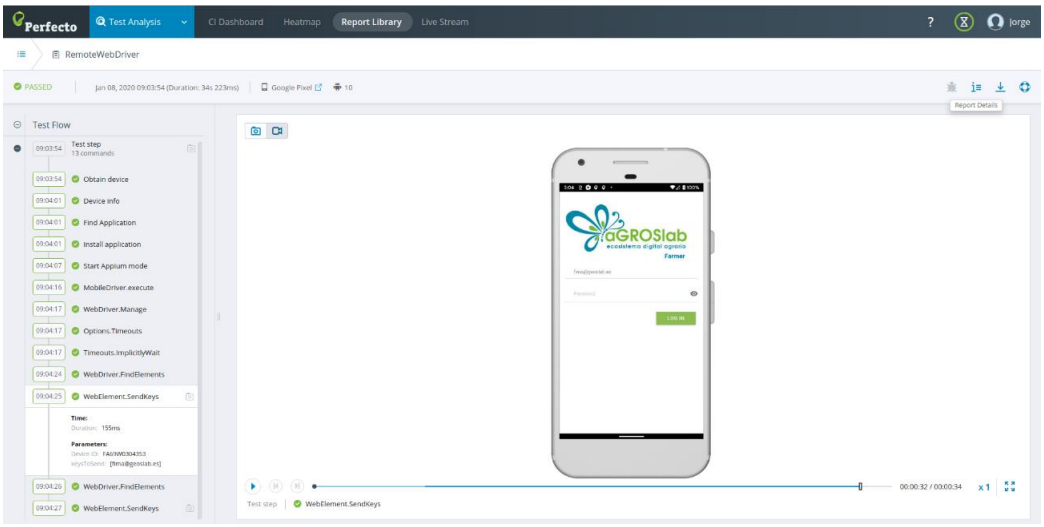

Figura 29: Video de la prueba realizada con Perfecto Mobile.

En la figura 29 aparece una imagen de la interfaz de Perfecto Mobile indicando el resultado de la prueba realizada en la aplicación.

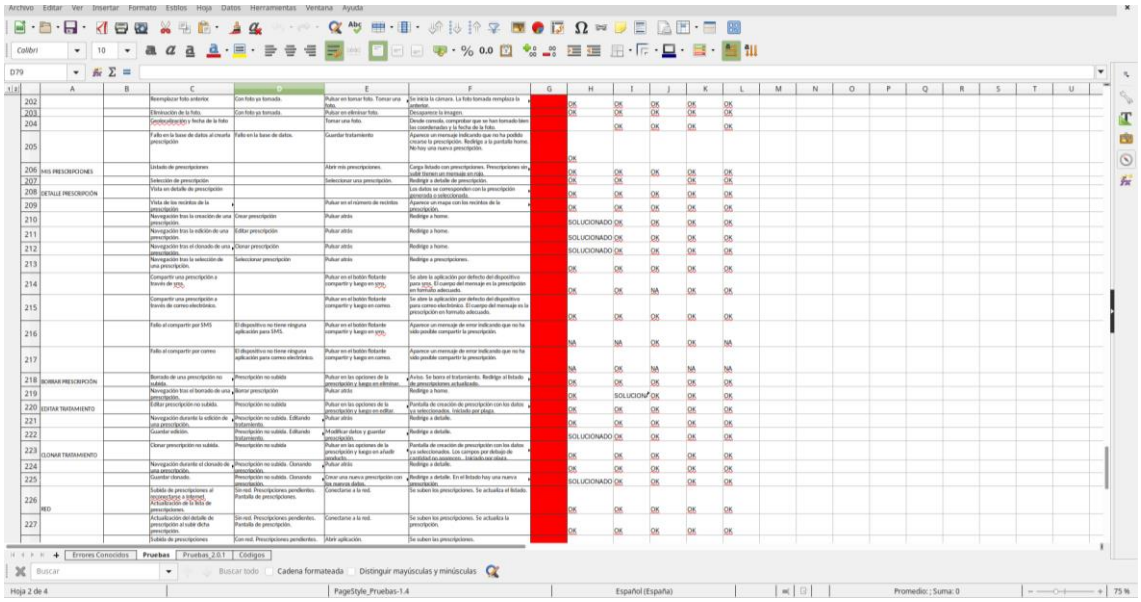

Figura 30: Recopilación de los casos de uso probados durante las pruebas manuales.

En esta última figura se muestra el Excel utilizado para la realización de las pruebas manuales. En el lado izquierdo aparecen los detalles de cada prueba y en el derecho el resultado de la prueba en cada dispositivo.