# Desarrollo de una solución para la delimitación de fincas, parcelas y siembras para su visualización en Google Maps

**Nombre Estudiante:** Jose Caselles Sanchez
Máster Universitario en Desarrollo de Aplicaciones para Dispositivos Móviles

**Nombre Consultor/a:** Eduard Martin Lineros
**Profesor/a responsable de la asignatura:** Carles Garrigues Olivella

07/06/2017

© Jose Caselles Sanchez
Reservados todos los derechos. Está prohibido la reproducción total o parcial de esta obra por cualquier medio o procedimiento, comprendidos la impresión, la reprografía, el microfilme, el tratamiento informático o cualquier otro sistema, así como la distribución de ejemplares mediante alquiler y préstamo, sin la autorización escrita del autor o de los límites que autorice la Ley de Propiedad Intelectual.

**FICHA DEL TRABAJO FINAL**

| | |
|---|---|
| **Título del trabajo:** | *Desarrollo de una solución para la delimitación de fincas, parcelas y siembras para su visualización en Google Maps* |
| **Nombre del autor:** | *Jose Caselles Sanchez* |
| **Nombre del consultor/a:** | *Eduard Martin Lineros* |
| **Nombre del PRA:** | *Carles Garrigues Olivella* |
| **Fecha de entrega (mm/aaaa):** | 06/2017 |
| **Titulación::** | Máster Universitario en Desarrollo de Aplicaciones para Dispositivos Móviles |
| **Idioma del trabajo:** | *Castellano* |
| **Palabras clave** | *Hortofrutícola, plantación , SIG* |

**Resumen del Trabajo (máximo 250 palabras):** *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*


La finalidad del trabajo es implementar una solución para registrar partes de plantación y que en estos partes de plantación se especifique las coordenadas para saber la superficie que se ha plantado. El uso de esta aplicación es para empresas hortofrutícola.

Se ha utilizado una metodología en cascada, donde se ha tenido una primera fase donde se han recogido los requisitos de la aplicación y de cuáles son los requerimientos del cliente. Seguido de la fase de análisis, donde se diseña el sistema, su estructura y prototipos. Después de la fase de análisis le sigue la fase de implementación, donde se ha desarrollado la solución propuesta, Y una última fase de pruebas para comprobar que la solución funciona correctamente.

Como resultado del trabajo, se ha obtenido una aplicación Android donde se realizan los partes de plantación y un módulo web el cual envía los datos que necesita la aplicación Android para poder realizar partes y la recepción de los partes hechos en los dispositivos Android.

La conclusión después de realizar el trabajo, es que en el ámbito agrícola se tienen muchas oportunidades para desarrollar nuevas aplicaciones móviles para mejorar la eficiencia de procesos productivos y la simplificación de tareas.

# Índice

1. Introducción ..... 1
1.1 Contexto y justificación del Trabajo ..... 1
1.3 Enfoque y método seguido ..... 3
1.4 Planificación del Trabajo ..... 3
1.5 Breve sumario de productos obtenidos ..... 6
1.6 Breve descripción de los otros capítulos de la memoria ..... 6
2. Análisis ..... 7
Análisis de los usuarios ..... 7
Encargado de plantación ..... 7
Departamento de planos ..... 8
Gerencia ..... 8
Prototipos ..... 9
Prototipo grafico de la aplicación en Android ..... 9
Prototipo del módulo web ..... 13
Casos de uso ..... 15
Modelo de datos ..... 17
Arquitectura del sistema ..... 19
3. Implementación ..... 20
4. Conclusiones ..... 33
5. Glosario ..... 35
6. Bibliografía ..... 38
7. Anexos ..... 39
Instrucciones para la compilación de los proyectos ..... 39
Proyecto Web ..... 39
Proyecto Android ..... 41
Manual de usuario ..... 42
Modulo web ..... 42
Aplicación Android ..... 48
Instrucciones para la instalación de las aplicaciones ..... 51

## Lista de figuras

**No se encuentran elementos de tabla de ilustraciones.**

# 1. Introducción

## 1.1 Contexto y justificación del Trabajo

En la empresa en la que trabajo, empresa hortofrutícola, pretende que los datos de plantación sean lo más exactos posibles. Actualmente se tiene un pequeño departamento el cual se dedica a realizar los mapas de las fincas, con sus parcelas y sus siembras. Este departamento diseña los planos en función de mapas y de datos de plantación que se le dan. Pero esta forma de trabajar hace que tengamos dos problemas, el primero de todos y que es poco frecuente, es que no concuerda el mapa de las siembras de finca con la realidad. Y el otro problema, que aparece con mayor frecuencia, es que la información que aparece en el mapa con las plantas plantadas en una siembra no concuerda con el de los partes de plantación.
Otro problema que no depende del departamento que realiza los mapas, es que en algunas fincas la plantación de las siembras no es la más óptima y esto causa que a veces para recolectar una siembra se tenga desperdiciar parte de la plantación de otra siembra para poder llegar con las cosechadoras a la siembra que se pretende recolectar.

Con esta solución se pretende tener las coordenadas que delimitan las fincas, las parcelas de estas y las siembras en el ERP de la empresa. Al tener estos datos en el ERP y al poder mostrar estas delimitaciones en Google Maps, los encargados de plantación podrían planificar mejor las siembras. Gerencia también podría ver esta información en cualquier momento a través del ERP sin tener que pedir al departamento de mapas esta información.
Otro beneficio que se espera obtener de esta solución es el de que la información de las siembras sea el correcto, número de plantas plantadas, la variedad de semilla plantada. Y simplificar el registro de partes de plantación en comparación a como se están realizando en este momento.
Por último, el registro de estos datos, permitirá que en el futuro desarrollo de una aplicación para registrar partes de recolección, esta pueda recuperar parte de esta información automáticamente mediante el uso de las coordenadas del dispositivo donde esté realizando el parte de recolección.

## 1.2 Objetivos del Trabajo

- Guardar en el sistema las coordenadas que delimitan una finca.
- Guardar en el sistema las coordenadas que delimitan las parcelas de una finca.
- Obtener el área de una finca.
- Obtener el área de una parcela de una finca.
- Registrar cuando se realizó y por quien, la delimitación de una finca o una parcela.
- Se debe poder modificar una delimitación de una finca/parcela y llevar un historial de versiones de estas modificaciones.
- Guardar en el sistema las coordenadas que delimitan una siembra.
- Al registrar las coordenadas de una siembra se deben indicar los siguientes datos:
    - Numero de ríos.
    - Variedad de semilla plantada.
    - Número de plantas plantadas.
    - Marco de plantación.
    - Fecha de plantación.
    - Equipo que ha realizado la plantación.
    - Hora de inicio y fin de plantación.
    - Horas trabajadas.
- Desarrollar un módulo web para el ERP, para poder visualizar las fincas, parcelas y siembras utilizando Google Maps.
- La aplicación móvil debe funcionar sin cobertura ya que algunas fincas no tienen cobertura móvil.
- Los datos se deben sincronizar automáticamente sin que el usuario tenga que estar atento a esta tarea.
- Se debe poder ejecutar tanto en un Tablet como en un Smartphone.
- La comunicación entre la aplicación móvil y el ERP será mediante un servidor que contiene servicios web.
- El intercambio de información entre la aplicación móvil y los servicios web será mediante JSON.
- La aplicación solo la pueden ejecutar los usuarios con permiso.
- Si el usuario cambia de dispositivo este debe de recuperar los datos de las siembras que ha registrado.
- El usuario que está en una finca debe disponer de todos los datos de esta.
- El registro de una plantación puede que se quede a medias y se debe poder recuperar los datos registrados para terminar con el registro.
- Una vez registrado los datos de una plantación, se deben poder modificar.
- La aplicación debe recuperar los datos de los equipos de trabajo y de los trabajadores que lo confieren.
- La aplicación móvil se desarrollara en Android por utilizarse Tablet con Android.
- El modulo web y los servicios web se desarrollaran en Java ya que el ERP de la empresa está desarrollado en Java.
- Los datos del sistema se guardan en una base de datos Oracle.

## 1.3 Enfoque y método seguido

Se ha decantado por el desarrollo de la aplicación móvil, de los servicios web y del módulo web por tener que integrar esta solución en el ERP de la empresa. Se han valorado algunas soluciones que existen en el mercado como son: *Campogest, Agroptima, Toolagro-mobile, Visual 3.0 de visualnacert*. Estas soluciones se han descartado ya que ninguna de ellas cumple con los requisitos y alguna de ellas se deben implantar con su propio ERP. Así que sería más costoso modificar alguna de estas soluciones, que el desarrollar una solución que cumpla con los requisitos y que se integre con el ERP de la empresa.

## 1.4 Planificación del Trabajo

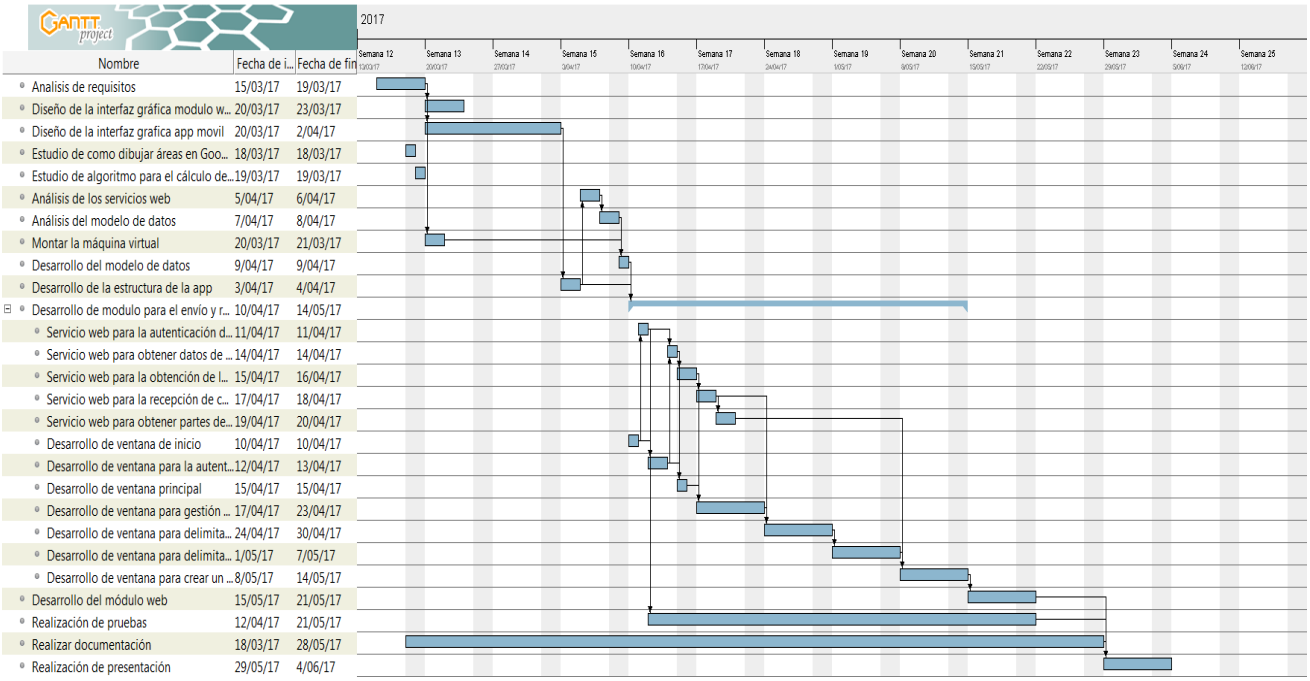
Para poder llevar a cabo el trabajo, se necesita un ordenador para el desarrollo de la aplicación móvil, los servicios web y el modulo web. Este ordenador debe disponer de un entorno de desarrollo para poder programar en Android ya que es la plataforma elegida sobre la que debe ejecutarse la aplicación. El entorno de desarrollo elegido para programar en Android es Android Studio, ya que es gratuito y es el entorno que se conoce. El entorno de desarrollo para los servicios web y el modulo web es Netbeans. La elección de este entorno es debido a que es un entorno para programar en Java, es gratuito y es el entorno que se utiliza en la empresa para los desarrollos en Java. El ordenador de desarrollo debe ser un portátil, para que cuando se realicen las pruebas en campo, si ale algún error se pueda depurar en el mismo momento.
Por otra parte necesitamos un servidor de aplicaciones, que contendrá los servicios web y el modulo web. El servidor de aplicaciones a utilizar será WildFly, por ser el que se utiliza en la empresa. Además del servidor de aplicaciones, se necesitara un servidor de base de datos Oracle para guardar los datos. Se utilizara Oracle Express Edition, ya que en la empresa se utiliza Oracle y esta versión es gratuita y es más que suficiente para realizar las pruebas y tener los script de creación de tablas, triggers, vistas, etc. De esta forma cuando se ponga en producción la aplicación no se deberán modificar estos scripts y se podrán ejecutar directamente en la base de datos de producción.
Tanto el servidor de aplicaciones como el servidor de base de datos se pueden tener en el mismo ordenador de desarrollo pero para hacer pruebas realistas, se instalaran en un ordenador distinto, en concreto en una máquina virtual Linux, a la que se tendrá acceso desde Internet las 24 horas. Así se podrán hacer pruebas en el campo. De esta forma se tiene un entorno de pruebas lo más parecido al entorno de producción en el que se ejecutara la aplicación tanto móvil como web.
Y para finalizar con los recursos necesarios, necesitaremos una Tablet que disponga de GPS y una ranura para ponerle una tarjeta sim 3g.

Las tareas a realizar junto con su estimación, son las siguientes:

- Análisis de los requisitos. 6 horas.
- Diseño de las interfaces gráficas.
    - Diseño de la interfaz gráfica de la app móvil. 18 horas.
    - Diseño de la interfaz gráfica del módulo web. 5 horas.
- Estudio de como dibujar áreas en Google Maps. 4 horas.
- Estudio de algoritmo para el cálculo de áreas mediante coordenadas. 4 horas.
- Análisis de los servicios web. 5 horas.
- Análisis del modelo de datos. 6 horas.
- Montar la máquina virtual con el servidor de base de datos y de aplicaciones. 4 horas.
- Desarrollo del modelo de datos. 4 horas.
- Desarrollo app móvil.
    - Desarrollo de la estructura de la app con todas las ventanas. 4 horas.
    - Desarrollo de ventana de inicio. 2 horas.
    - Desarrollo de ventana para la autenticación de usuario. 2 horas.
    - Desarrollo de ventana principal. 4 horas.
    - Desarrollo de ventana para gestión de fincas. 16 horas.
    - Desarrollo de ventana para delimitar una finca. 16 horas.
    - Desarrollo de ventana para delimitar parcelas de una finca. 16 horas.
    - Desarrollo de ventana para crear un parte de plantación y delimitar el área plantada. 16 horas.
    - Desarrollo de modulo para el envío y recepción de datos. 8 horas.
- Desarrollo de servicios web.
    - Servicio web para la autenticación de usuario autorizados para ejecutar la aplicación móvil. 2 horas.
    - Servicio web para la obtención de los datos de fincas. 2 horas.
    - Servicio web para obtener datos de equipos. 2 horas.
    - Servicio web para obtener partes de plantación. 2 horas.
    - Servicio web para la recepción de coordenadas de una finca, sus parcelas y sus siembras. 4 horas.
- Desarrollo del módulo web para visualizar en Google Maps las fincas, las parcelas y sus siembras. 16 horas.
- Realización de pruebas. 30 horas.
- Realizar documentación. 60 horas.
- Realización de presentación. 16 horas.

La estimación de horas totales para llevar a cabo el trabajo es de 274 horas. Se pretende trabajar una media de 3 horas al día de lunes a viernes y una media de 5 horas al día los sábados y domingos.

Diagrama de Gantt con la planificación de las tareas.




Los hitos planificados para cada PEC son los siguientes:

1. PEC2:
    - Diseño de las interfaces graficas tanto de la aplicación móvil como del módulo web.
    - Dibujar áreas en Google Maps.
    - Algoritmo para calcular áreas mediante una serie de coordenadas.
    - Disponer de la máquina virtual con el servidor de base de datos y de aplicaciones.
    - Prototipo de la aplicación móvil.
    - Documentación de lo desarrollado.

2. PEC3:
    - Modelo de datos
    - Implementación de los servicios web.
    - Implementación de las ventanas de la aplicación móvil.
    - Implementación del módulo web.
    - Realización de pruebas.
    - Documentación de lo desarrollado hasta la fecha.

3. PEC4:
    - Depuración de algún error en la aplicación.
    - Terminar la documentación del trabajo.
    - Manual de usuario.
    - Manual de instalación.
    - Realizar la presentación.

## 1.5 Breve sumario de productos obtenidos

Se obtendrá una aplicación para dispositivos móviles, además se necesitara desarrollar los servicios web necesarios para guardar los datos que se registran con la aplicación en la base de datos del ERP de la empresa.
Y por último, se desarrollara un módulo web que se incluirá en el ERP de la empresa, para ver las fincas, parcelas y siembras en Google Maps.

## 1.6 Breve descripción de los otros capítulos de la memoria

El siguiente capítulo ira destinado al análisis, se analiza con más profundidad el problema a resolver y como se va a dar solución. Viendo cuales van a ser todas las partes que van a formar la solución y cómo van a interactuar entre ellas. Se diseñaran varias interfaces de usuario para encontrar el diseño más sencillo para el usuario.
Tendremos un capítulo sobre el estudio de como dibujar formas sobre Google Maps.
Y también tendremos un capitulo donde se explicara el algoritmo para el cálculo de áreas mediante una serie de coordenadas.
El siguiente capítulo se explicara el modelo de datos de la solución.
A continuación tendremos un capitulo explicando cómo se montan los servicios web y cuales son necesarios para el proyecto.
Y por último tendremos un capitulo donde se verá como se ha implementado la aplicación móvil.

# 2. Análisis

Una parte importante del análisis a la hora de desarrollar una aplicación es estudiar cuales van a ser los usuarios. En el desarrollo de una aplicación móvil además de estudiar cuales van a ser los usuarios, también es importante saber en qué condiciones va a usar la aplicación. Por tanto lo primer que vamos hacer es ver cuáles son los usuarios de la solución, cuál va a ser el contexto en que van a utilizarla y que es lo que esperan de ella.

## Análisis de los usuarios

Con los requisitos de la solución se ve claramente que existen tres usuarios bien diferenciados. Por una parte tenemos los encargados de un equipo de plantación que serán los usuarios de la aplicación móvil. Por otra parte están los usuarios del departamento de planos. Y por último tenemos a los usuarios pertenecientes a gerencia. Cada uno de estos usuarios tiene un perfil bien definido.

## Encargado de plantación

El encargado de plantación tiene una edad comprendida entre los 35 y 50 años. Por lo general residen en zonas rurales en medio del campo con pocos habitantes. Su formación académica es la mínima obligatoria, algunos solo tienen hasta la EGB.
En cuanto al conocimiento que tienen sobre dispositivos móviles, la mayoría tienen unos conocimientos muy bajos, por lo que la empresa sabe que deberá darles formación y durante unos días un soporte en el campo para ver que saben utilizar la aplicación correctamente.
Lo normal, es que utilicen la aplicación cuando estén trabajando en el campo, o por lo menos en el momento para delimitar la franja de parcela que se ha cultivado. Mientras que el resto del parte de plantación lo pueden terminar una vez que lleguen a sus casas o en cualquier otro sitio. Pero lo más frecuente, es que una vez terminada la plantación, realicen el parte de plantación en el mismo campo. Por esta razón hay que tener un especial cuidado al seleccionar los colores de la aplicación, para que se pueda ver todo correctamente tanto si hace un día muy soleado como muy nublado. Otro requisito a tener muy en cuenta, es que al ser un usuario que ha trabajado toda su vida en el campo, los dedos de estos son grandes. Así que hay que prestar atención en separar bien los icono y hacerlos grandes para que no tengan ningún tipo de problema al intentar pulsar un icono o cualquier elemento de la aplicación y que no se pulse cualquier otro componente.

## Departamento de planos

Los usuarios del departamento de planos tienen una edad comprendida entre los 25 y 40 años. Por lo general tienen un módulo superior de FP. A diferencia de los encargados de plantación, estos viven en pueblos con mayor número de habitantes entre 5,000 a 30,000.
Este usuario está acostumbrado a utilizar un ordenador en su día a día. Su conocimientos informáticos le permitirán utilizar el modulo web para delimitar las fincas sin ningún tipo de problema.
En este caso, trabajaran con la solución en la oficina, en un ambiente en el cual se tiene una buena iluminación y disponen de monitores con pantallas de 20 a 24 pulgadas.

## Gerencia

Por último, tenemos los usuarios de gerencia, los cuales tienen una edad comprendida entre los 45 y 60 años. Todos tienen una formación académica universitaria y viven en urbanizaciones residenciales. Los conocimientos informáticos son básicos pero están acostumbrados a utilizar aplicaciones de reporting como SAP BusinessObjects o QlikView.
Al igual que el departamento de planos, utilizaran el modulo web en la oficina, en un entorno donde la iluminación es la correcta, aunque con monitores no tan grandes, de unos 17 pulgadas. Algunos no tienen de ordenadores de sobremesa sino que utilizan portátiles, así que hay que tener atención a que el modulo web se pueda ver claramente en monitores más pequeños como puede ser de 13 pulgadas. Otro requisito relacionado con este punto, es que hay que comprobar que el modulo web funcione correctamente con el navegador de Mac OS X ya que alguno usuario de gerencia pose Mac en vez de un portátil con Windows.

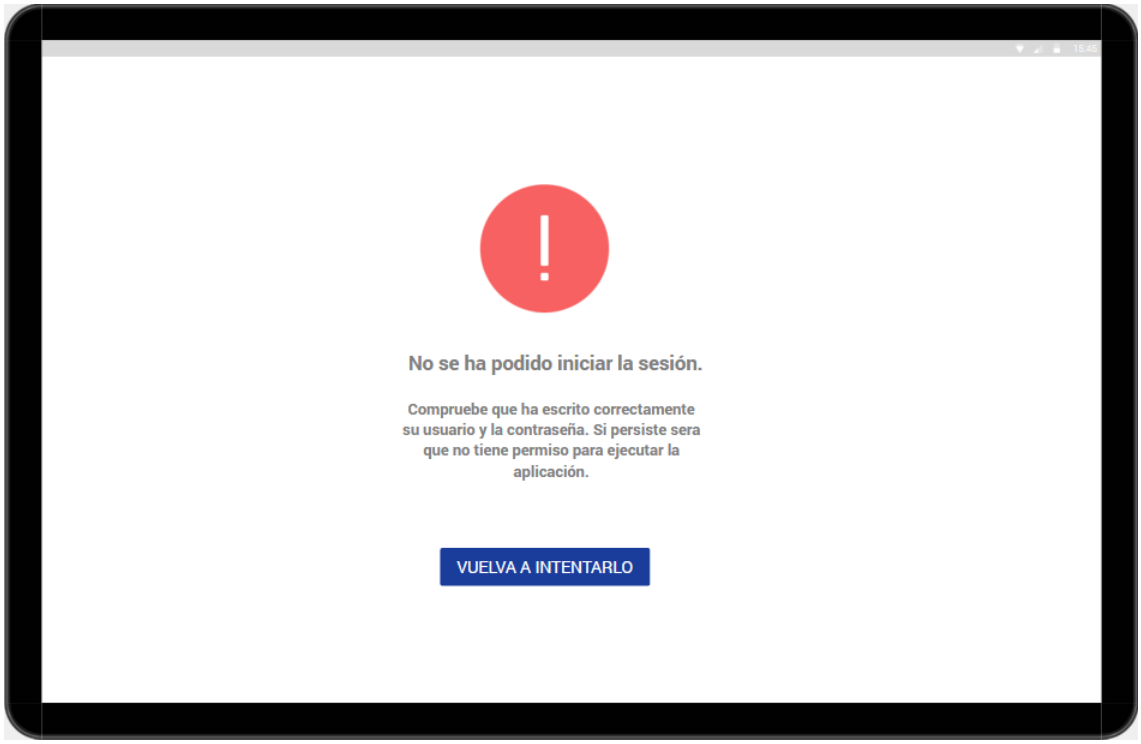
## Prototipos

Al estar dividida la solución en una aplicación Android y en un módulo web se han realizado dos prototipos para las interfaces gráficas. Se ha trabajado más en la solución Android ya que se debe desarrollar por completa la aplicación Android.

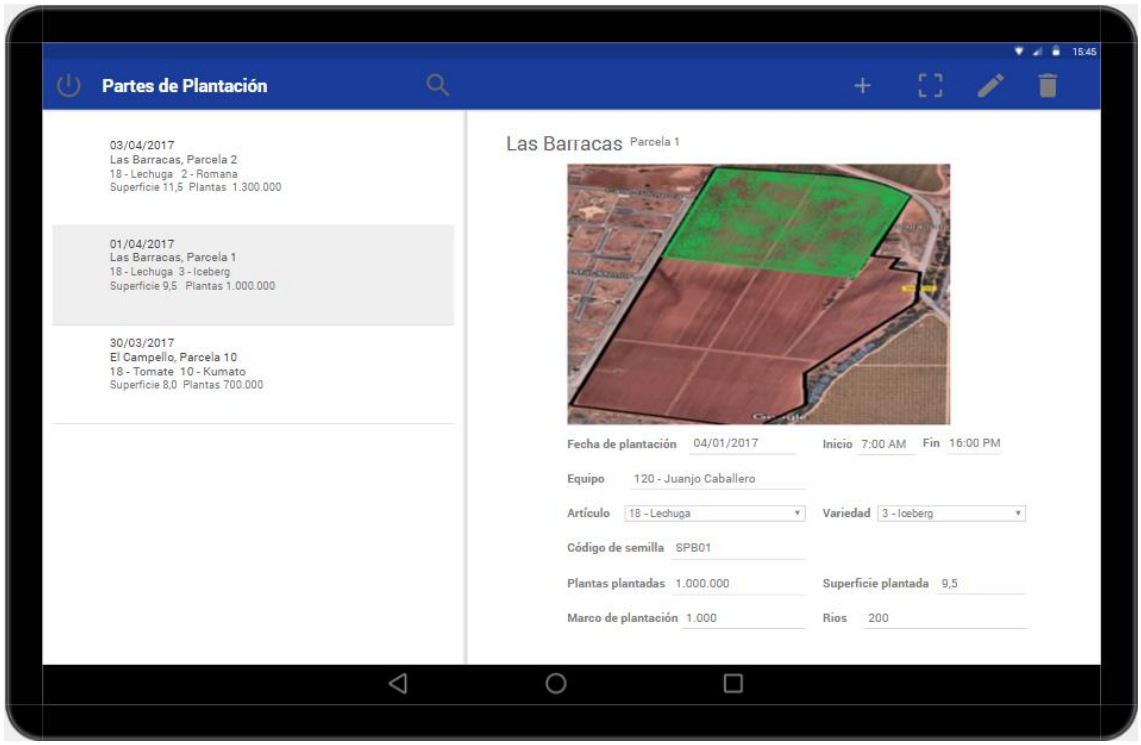
## Prototipo grafico de la aplicación en Android

Esta sería la ventana de inicio donde el usuario debe introducir su usuario y contraseña.

En el caso de que el usuario no esté autorizado a ejecutar la aplicación o introduzca mal los datos para iniciar sesión se redireccionara a la siguiente ventana.



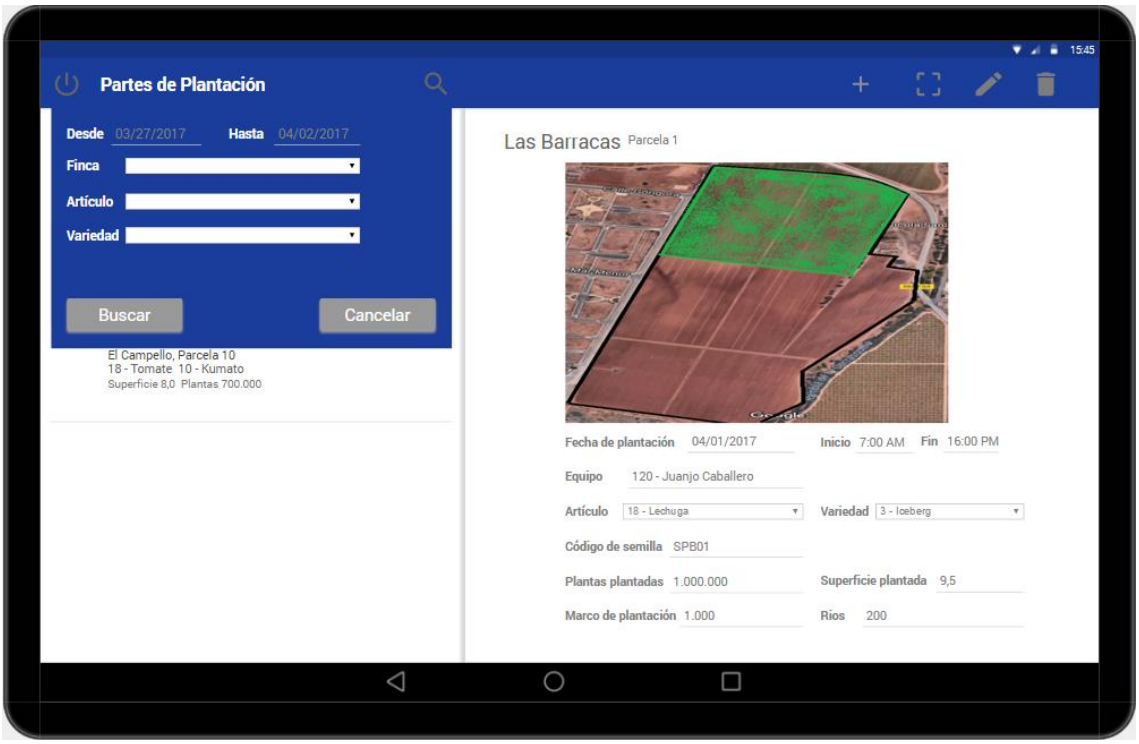

Y esta es la ventana que le aparece a un usuario que ha iniciado sesión.



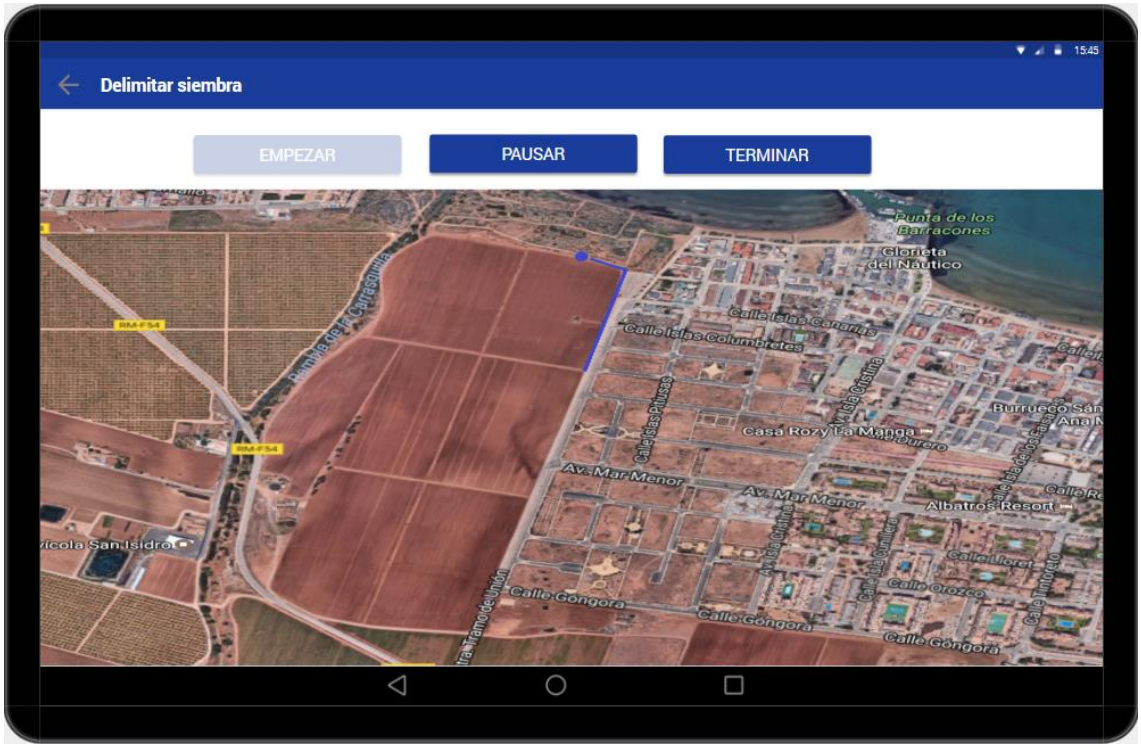
En la parte de la izquierda tenemos el listado de los partes de plantación del usuario. Y en la parte de la derecha aparecen los datos del parte de plantación seleccionado junto con la imagen de Google Maps delimitando la finca y la parte plantada de ese parte.
Y en la parte de arriba disponemos de las distintas opciones que dispone el usuario.
De izquierda a derecha estas son las opciones:

- Cerrar la sesión.
- Mostrar filtros de búsqueda de partes.
- Crear un nuevo parte de plantación.
- Delimitar el parte de plantación seleccionado.
- Editar el parte de plantación seleccionado.
- Eliminar el parte seleccionado.

Estos serían los filtros de búsqueda de partes.



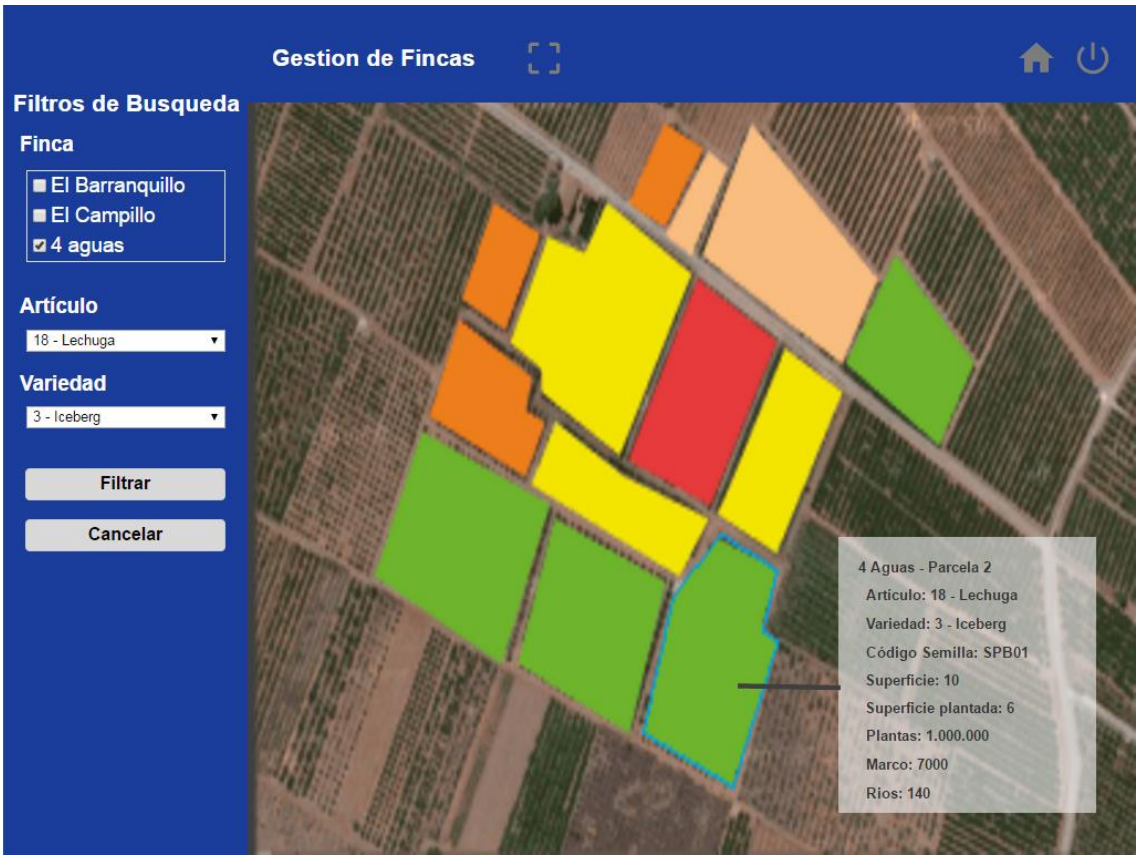
Y por último, esta sería la ventana durante el proceso de delimitar una siembra vinculada a un parte de plantación.




Tendremos tres botones un para indicar que se empieza el proceso para ir obteniendo las coordenadas del GPS e ir pintando la delimitación de la siembra. En este ejemplo podemos ver que aparece una línea azul y un punto del mismo color que indica la posición del encargado en la finca.
Tenemos un botón para pausar el proceso de delimitar la siembra. Y por último, tenemos la opción de terminar.

## Prototipo del módulo web

Esta sería el diseño de la interfaz gráfica del módulo web.

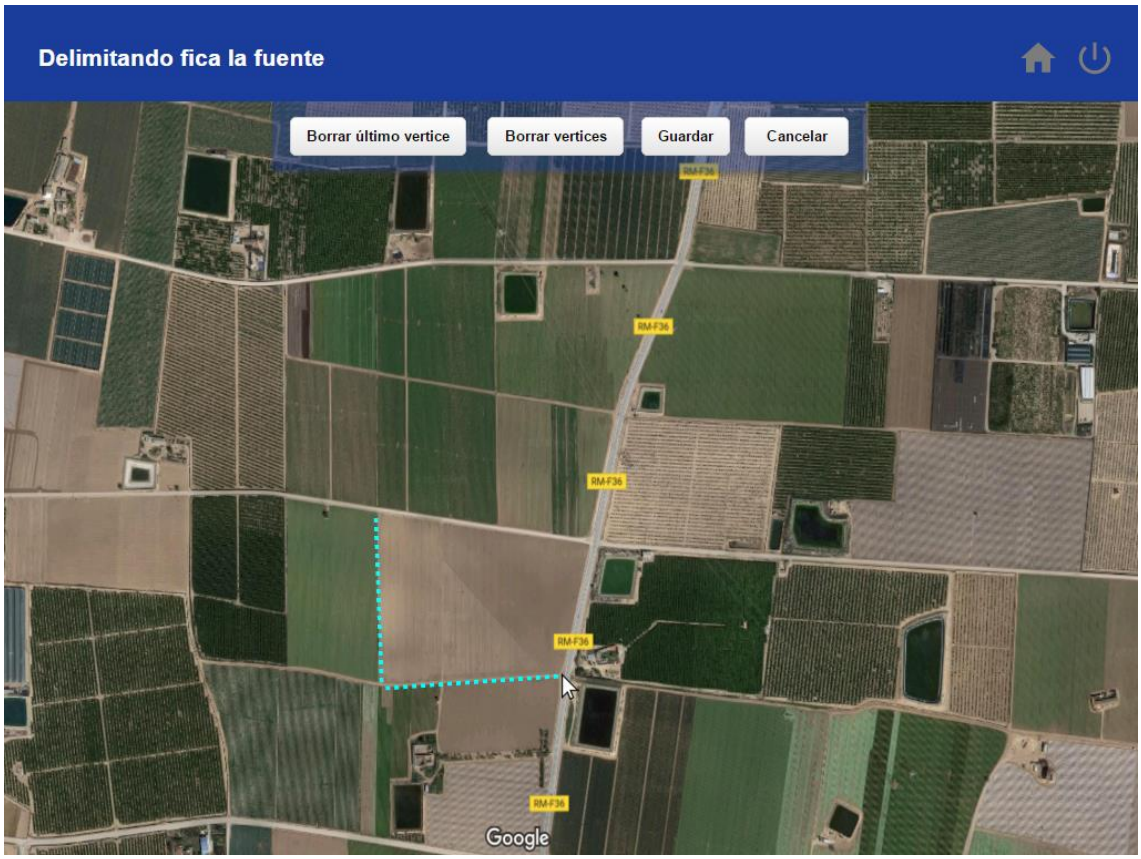



En la parte principal se mostraría el mapa de Google Maps, coloreando las parcelas de las fincas que cumplan los filtros de búsqueda. Si se hace clic en una de estas parcelas se mostrara un pequeño panel con la información de esta parcela.

En la parte de arriba tenemos las distintas opciones. La opción más importante y la que nos interesa es esta , que sirve para delimitar una finca y guardar las coordenadas de esta en la base de datos. Esta opción solo estará disponible para los usuarios del departamento de planos ya que gerencia no tiene que hacer este trabajo.

En la parte de la izquierda tenemos un panel que se puede ocultar y mostrar con los filtros de búsqueda para que solo aparezcan las fincas que cumplan con los criterios de búsqueda.

Y esta sería la vista durante el proceso de delimitar una finca por parte del departamento de planos.

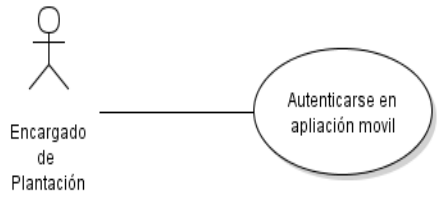



Durante el proceso, el usuario solo debe ir marcando los vértices para delimitar la finca mediante clics de ratón sobre el mapa. En la imagen se puede ver como se han marcado tres vértices y se van uniendo mediante línea punteada para que se vea que se está delimitando correctamente la finca.
Durante este proceso aparece un panel con las distintas opciones que dispone el usuario para poder delimitar una finca. La primera de las opciones es la que permite eliminar el último vértice marcado por si se ha equivocado durante el proceso. Si se quiere empezar el proceso desde el principio, el usuario tiene la opción de eliminar todos los vértices. La opción de guardar, guarda los vértices que el usuario ha marcado sobre el mapa. En este punto, la aplicación debe comprobar que los vértices que se han marcado en el mapa representan una figura cerrada. Y por último, se tiene la opción de cancelar el proceso con lo cual no se guardara nada de lo hecho sobre el mapa.

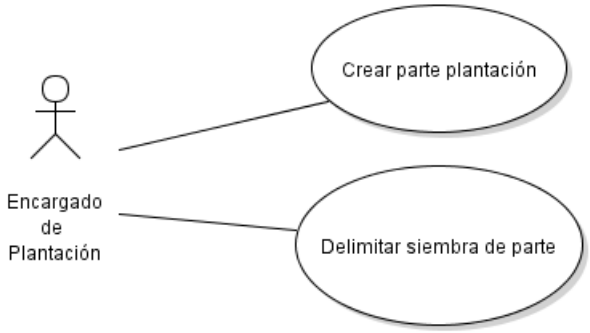
## Casos de uso

Estos son los casos de usos que se han identificado a partir de los requisitos de la solución.

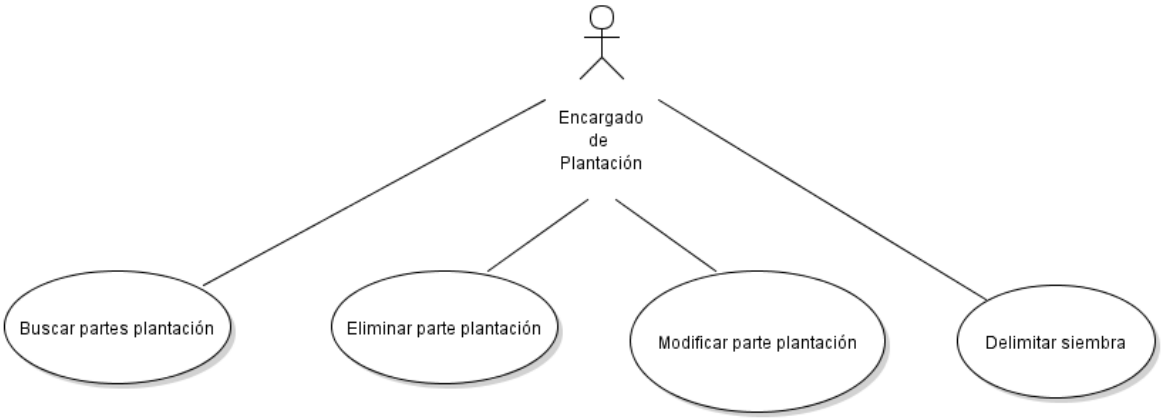
El primer caso de uso que detectamos es el que el encargado de plantación debe autenticarse para acceder a la aplicación móvil. En este proceso además de que el los datos del usuario son correctos, este debe tener el permiso para poder ejecutarla.



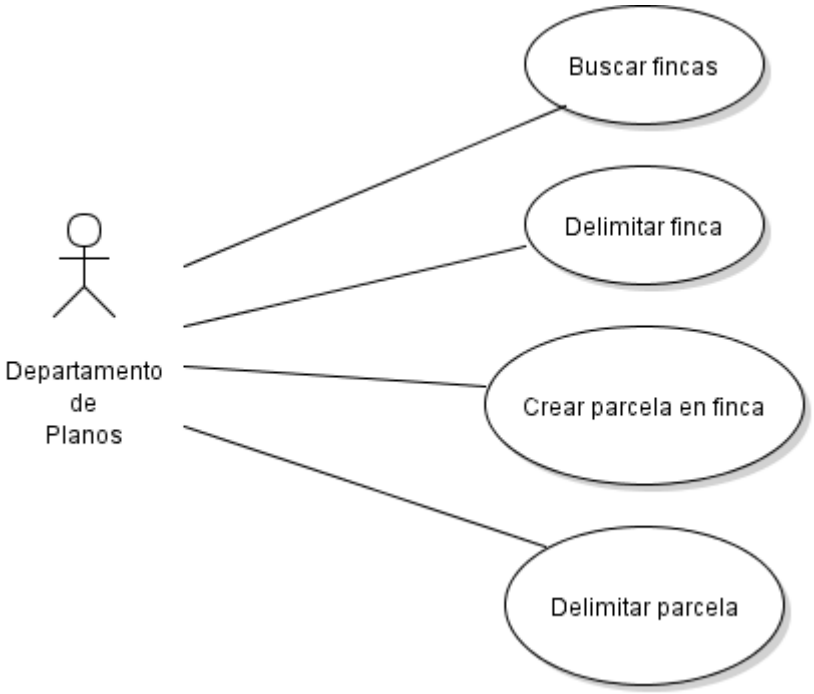

Los siguientes casos de uso que se detectan para este actor es el de dar de alta un parte de trabajo, y en este proceso también debe poder delimitar la siembra vinculada a este parte.



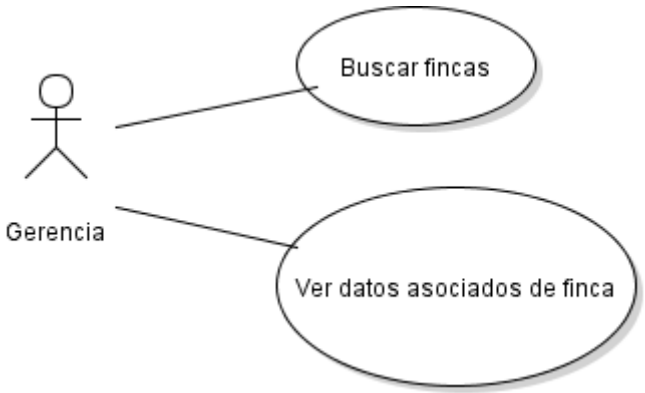

Y el último tenemos los siguientes casos de usos relacionados para este actor, es el que se define el proceso por el cual el actor puede buscar partes de plantación, eliminar un parte de plantación, modificar los datos de este y modificar la delimitación de una siembra de un parte.



El siguiente actor que encontramos es el departamento de planos. Los casos de uso que se han detectado para este actor son los de poder buscar fincas para verlas en el mapa de Google, delimitar estas fincas, dividir la finca en parcelas y delimitarlas.

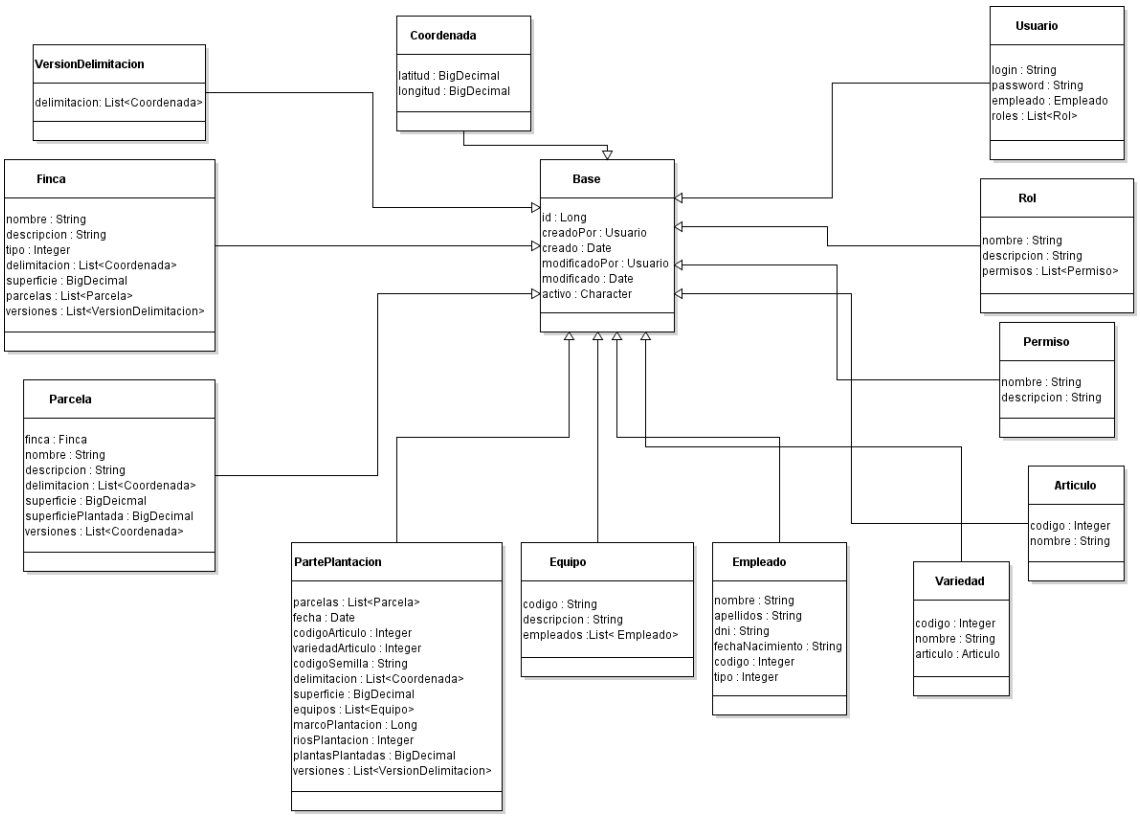



Por último, tenemos al actor que representa a los usuarios de gerencia. Este actor participa en dos casos de usos muy sencillos. El de buscar fincas para visualizarlas en el mapa de Google Maps y ver los datos de estas fincas, parcelas, partes de plantación, etc.



## Modelo de datos

Con los requisitos y después de hacer el análisis esta sería el diagrama con el modelo de datos, de las entidades que se deben guardar en base de datos.

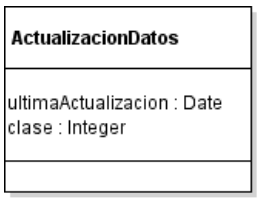


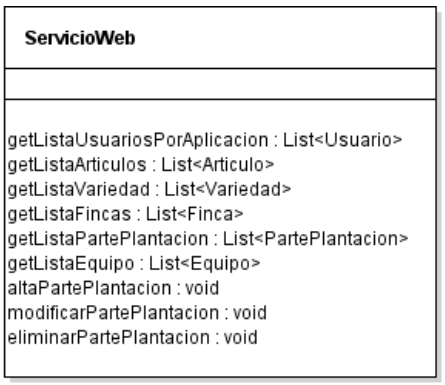

Todas las clases extienden de la clase *Base* que contiene los datos que todas las clases deben tener. A continuación se describe las clases:

- La clase *Usuario*, representa un usuario del sistema, el cual contiene el nombre y la contraseña para poder entrar al sistema. Este usuario puede tener uno o más roles para definir los permisos que tiene este.
- La clase *Rol*, representa un rol del sistema, el cual tiene una serie de permisos que definen las cosas que puede hacer en el sistema.
- La clase *Permiso*, representa un permiso.
- La clase *Articulo*, representa los distintos tipos de artículos.
- La clase *Variedad*, representa las distintas variedades que tiene un artículo.
- La clase *Equipo*, representa un equipo de trabajadores.
- La clase *Empleado*, representa un empleado de la empresa.
- La clase *Coordenada*, representa una coordenada geográfica para definir una posición en un mapa.
- La clase *Finca*, representa una finca donde se cultiva. Esta está formada por una serie de coordenadas para delimitarla en un mapa. Y además está compuesta por una serie de parcelas.

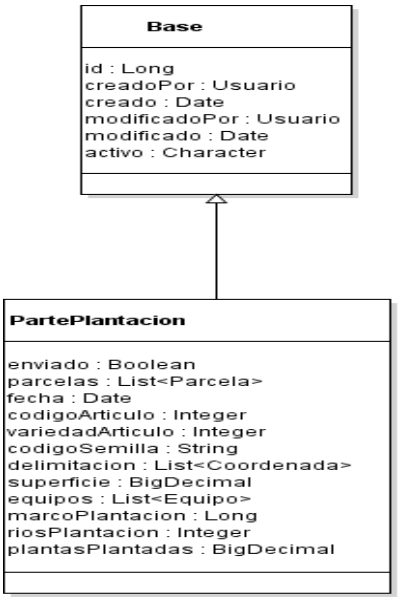
- La clase *Parcela*, representa una división física de una parcela. Está delimitada por una lista de coordenadas.
- La clase *PartePartacion*, representa un parte de plantación.
- La clase *VersionDelimitacion*, representa una versión de la delimitación tanto de una finca, parcela o una siembra de un parte de plantación. Así se puede ver si se modifican estos datos frecuentemente o no.

Las siguientes clases representa la clase que contiene los servicios web para la comunicación entre la aplicación Android y la base de datos del sistema. La clase *ActualizacionDatos* nos sirve para saber cuál es la fecha de la última actualización de un tipo de objeto como puede ser *Finca, Equipo,* etc, para saber si se tiene que descargar el listado de ese tipo de objetos.



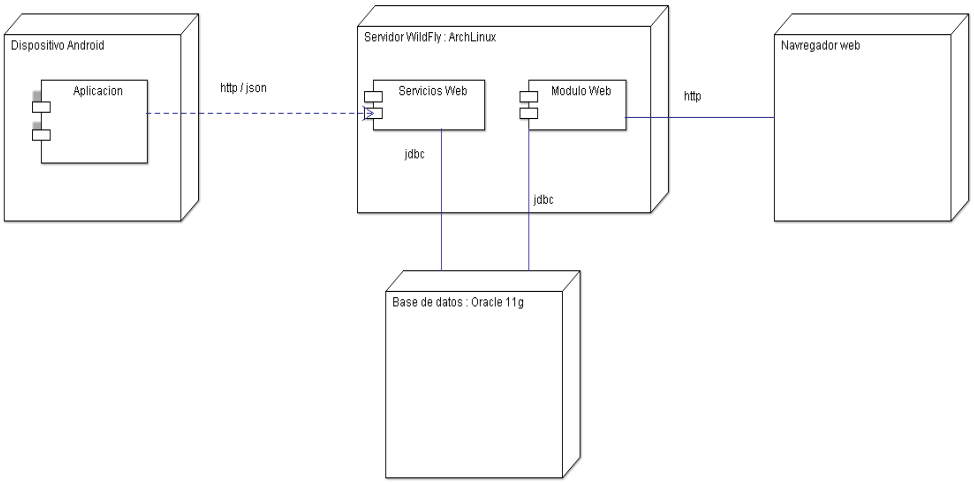




Por último se debe modificar la clase *PartePlantacion* en la aplicación Android para saber si ese parte se ha enviado o no al servidor. Esta clase contiene un atributo de tipo *Boolean*, el cual tendrá el valor de *true* si se ha enviado y se ha recibido correctamente o *false* en caso contrario.



## Arquitectura del sistema

Este es el diagrama de despliegue. Como se puede ver tenemos que en los dispositivos móviles Android se tiene la aplicación Android. Esta se comunica a través de peticiones http a la aplicación web que contiene los servicios web en el servidor de aplicaciones WildFly. Este, a su vez, devuelve los datos a través de una respuesta http pero los datos están formateados en formato JSON. El servidor de aplicación WildFly se comunica con el servidor de base de datos Oracle que está en otra máquina mediante JDBC.
Por otra parte en el mismo servidor de aplicaciones, tendremos instalado el modulo web, el cual los usuarios accederán a este mediante navegadores web. Este módulo como la aplicación de los servicios web se comunica con la base de datos mediante JDBC.



# 3. Implementación

Para empezar con esta parte del trabajo, se tuvo que preparar un ordenador para poder cometer el trabajo. Para la programación de la parte web se instaló la JDK 1.8, la herramienta maven y el entorno de desarrollo de programación Netbeans 8.1. Por otra parte, para la programación de la aplicación Android, se instaló el entorno de desarrollo Android Studio 2.3. Y para poder realizar las pruebas, se instaló una máquina virtual para contener una base de datos Oracle 11g Express, y se instaló un servidor de aplicaciones WildFly 10. De esta forma, las pruebas se realizarían en un entorno lo más parecido al entorno que se tiene en producción.

Una vez montado el entorno de programación como el de pruebas se siguió con el estudio de cómo trabajar con el servicio de Google Maps. Para poder trabajar con este servicio se debe tener una licencia con una clave, por tanto lo primero que se tuvo que hacer es registrar el proyecto en https://console.developers.google.com para que pueda utilizar las API de Google. En nuestro caso registramos una API Key. Antes de seguir con el estudio de la utilización de Google Maps JavaScript API, que es el API que se debe utilizar para poder incorporar y trabajar con un mapa de Google en una página web, se comprobó que no existiese ningún componente del framework de Prime Faces que fuese para trabajar con un mapa de Google. La decisión de hacer esto es porque el ERP de la empresa utiliza este framework para el desarrollo de páginas web, el cual facilita el desarrollo de estas, y se quiere que este nuevo módulo tenga la misma estética y estilo que el resto del ERP. No se quiere tener un módulo que parezca que es un añadido y ajeno al ERP.

Se comprobó que el framework Prime Faces dispone de un componente llamado Gmap, el cual nos permite trabajar de forma más sencilla que el API que nos facilita Google. Se hicieron varias pruebas para comprobar que este componente se podía utilizar tal como se quería en el proyecto y para ello si hicieron varias ventanas de pruebas.

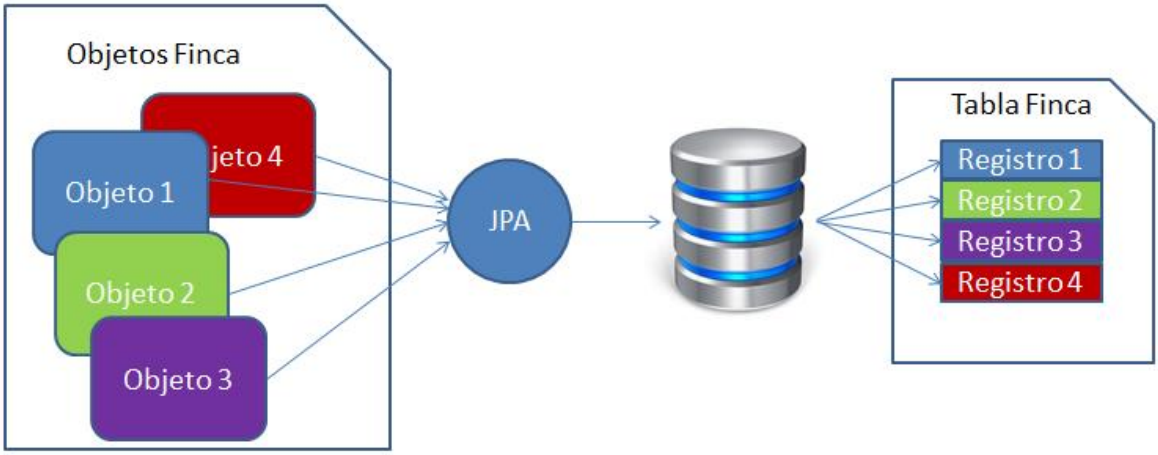
Una vez visto que se iba a utilizar el componente Gmap de Prime Faces, se siguió con la implementación del modelo de datos. Para no mezclar el proyecto web el que contendría el modulo web con las páginas web y su programación, se creó un proyecto aparte solo con el modelo de datos. Y este proyecto se incluiría como librería en el proyecto web.

Este pequeño proyecto se nombró como Modelo PlantacionDroid y además de contener el modelo de datos contiene el código para el acceso de base de datos. En Java se tiene el API, JPA, que permite vincular un una clase con una tabla de una base de datos. De forma que los registros de una tabla se convierten en objetos de la clase vinculada a esta tabla. Así, el programador no debe crear clases intermedias para transformar los datos de un objeto a registros de base de datos y viceversa.
JPA es solo una especificación, así que se debe utilizar una implementación. La implementación seleccionada es Eclipselink que es la implementación que se utiliza en el ERP.

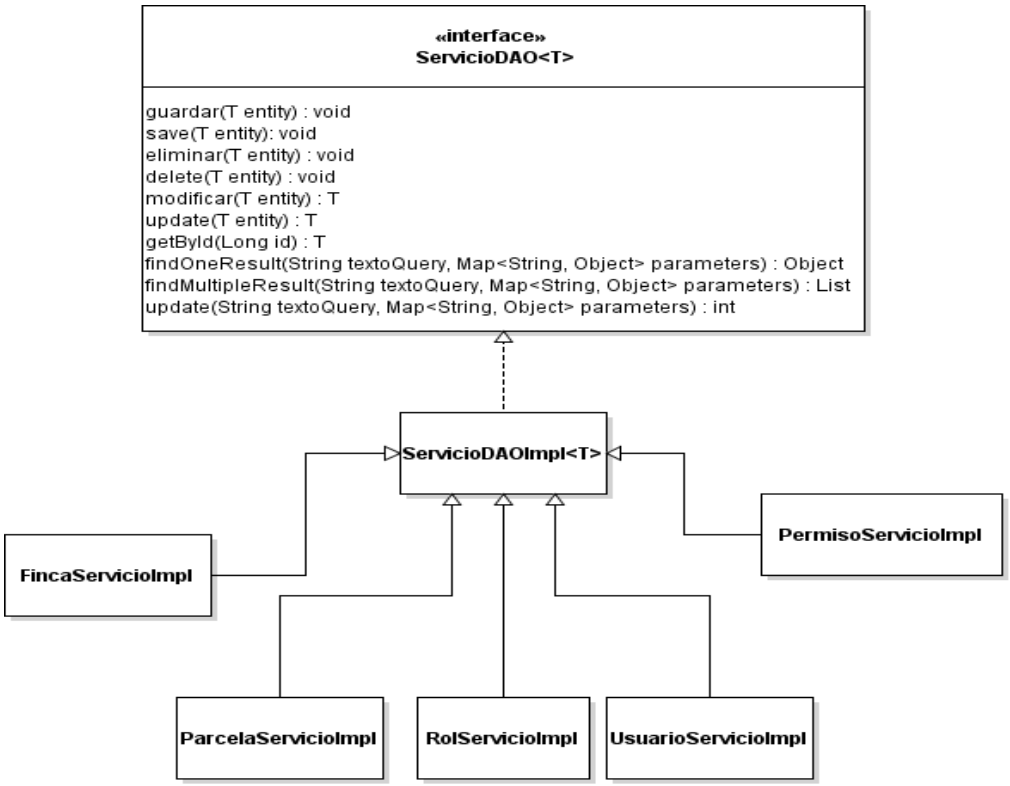
Además de desarrollar las clases del modelo de datos se implementaron las clases para realizar las consultas, las inserciones, modificaciones de los datos.

A continuación se muestra una clase que representa una *Variedad* y que esta mapeada con la tabla de base de datos llamada *VARIEDAD*:

```
@Entity
@Table(name = "VARIEDAD")
@SequenceGenerator(name = "VARIEDAD_SEQ", sequenceName = "VARIEDAD_SEQ", allocationSize = 1)
public class Variedad extends Base implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @Column(name = "ID_VARIEDAD")
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "VARIEDAD_SEQ")
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "ID_ARTICULO")
    private Articulo articulo;

    @Column(name = "CODIGO")
    private Integer codigo;

    @Column(name = "NOMBRE")
    private String nombre;

}
```



Por cada clase se ha especificado una interfaz y se ha implementado un servicio encargado de recuperar, guardar, modificar y eliminar los datos en base de datos para esa clase. El servicio implementa la clase *ServicioDAOImpl* que contiene los métodos básicos para todas estas tareas. De esta forma, en las implementaciones de los servicios, lo único que debemos realizar es la implementación para realizar las consultas en la base de datos. Las consultas se realizan mediante JPQL que es una mezcla de SQL con objetos.



A continuación vemos como el servicio para el objeto *Variedad* se implementa un método para obtener todas las variedades y otro método para obtener las variedades que pertenecen a un *Articulo* determinado.

```
@Dependent
public class VariedadServicioImpl extends ServicioDAOImpl<Variedad> implements VariedadServicio {

    private static final Log LOGGER = LogFactory.getLog(VariedadServicioImpl.class);

    private static final long serialVersionUID = 1L;


    @Override
    public List<Variedad> listaVariedades() {

        List<Variedad> lista = null;

        try {
            String consulta = "Select v From Variedad v Where v.activo = 'S' "
                + " Order By v.codigo";
            lista = findMultipleResult(consulta, null);

        } catch(Exception e) {
            LOGGER.error(e);
        }

        listaNoNula(lista);

        return lista;
    }

    @Override
    public List<Variedad> listaVariedades(Articulo articulo) {

        List<Variedad> lista = null;

        if(articulo != null) {
            try {
                String consulta = "Select v From Variedad v Where v.activo = 'S' "
                                    + " and v.articulo = :articulo";
                Map<String, Object> parametros = new HashMap<>();
                parametros.put("articulo", articulo);

                consulta = consulta + " Order By v.codigo";

                lista = findMultipleResult(consulta, parametros);

            } catch(Exception e) {
                LOGGER.error(e);
            }
        }

        listaNoNula(lista);

        return lista;
    }

}
```

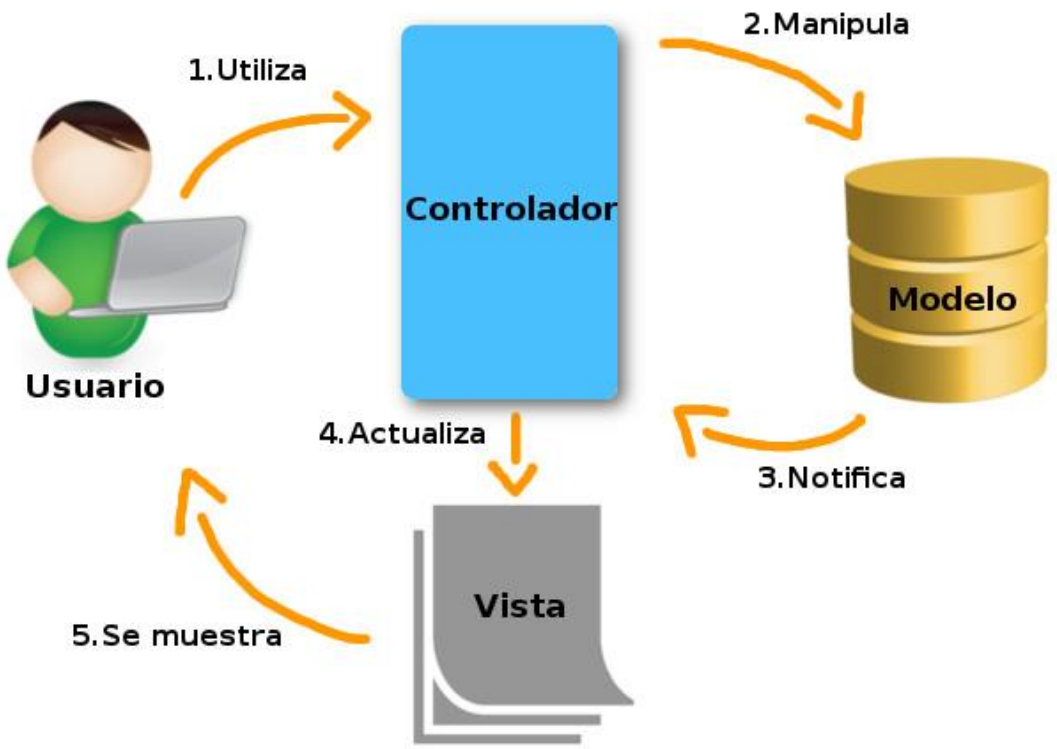
Una vez hecho el modelo de datos, se generaron las tablas en la base de datos.

Teniendo la librería con el modelo de datos, los servicios de estos y las tablas creadas en la base de datos, se empezó a desarrollar el modulo web. El modulo web, como se ha comentado, utiliza el framework Prime Faces que es una librería que contiene una biblioteca de componentes java para facilitar el desarrollo de e aplicaciones web en Java.

Este framework está desarrollado para ser usado con JSF. JSF es un framework de Java para el desarrollo de aplicaciones web, mediante el cual facilita el uso de una clase java para que controle una página.

Se ha escogido el usar JSF para el desarrollo web porque facilita mucho el desarrollo de aplicaciones web en Java. La utilización de este framework nos obliga utilizar el patrón de diseño modelo-vista-controlador. Que en mi opinión es la mejor practica en el desarrollo de aplicaciones web, ya que de esta forma tenemos por una parte la vista, el diseño de las páginas, que es independiente de los controladores, clases java que implementan la lógica y el comportamiento de las páginas. Las páginas web o ventanas están definidas en ficheros con la extensión xhtml. Y los controladores están definidos en clases java llamadas *beans*, que controlan estas páginas web. Y por último los servicios creados en el proyecto anterior para la lógica y las interacciones con el modelo de datos y la base de datos.

El siguiente esquema, muestra cómo funciona el modelo de díselo mvc, modelo-vista-controlador, en una página web.



Teniendo clara la estructura para el desarrollo del módulo web, se empezó con desarrollar la ventana de acceso al módulo y su controlador. De esta manera para realizar las pruebas no necesitaremos incorporar el modulo web en el ERP. Así que para el trabajo se ha añadido el código necesario para que se pueda ejecutar este módulo como si fuese una aplicación web independiente.

La página de acceso se llama *login.xhtml* la cual está controlada por el bean *LoginBean*. Esta página recibe el nombre de usuario y la contraseña de este y se lo pasa al bean para comprobar que existe un usuario con ese nombre y esa contraseña. Si es todo correcto entonces el bean redirecciona a la ventana *main.xhtml* que contiene los enlaces a las distintas ventanas del módulo.

A continuación se desarrolló el bean con el cual el usuario puede delimitar una finca mediante Google Maps. Este bean, es el bean más importante del módulo, es el controlador llamado *MapaGestionFincaBean*. Además de delimitar una finca, gestiona las parcelas de esta y la delimitación de estas.
Este bean se centra en el componente *Gmap* de Prime Faces. Este componente puede definirse para que esté atento a distintos eventos que se realizan sobre este mapa y que el controlador los capture y realice las acciones pertinentes. Hay que comentar que para poder utilizar este componente en una página xhtml debemos añadir la biblioteca de funciones JavaScript de Google junto con la clave que al empezar con la implementación se ha generado. Para poder utilizar esta biblioteca y el componente Gmap pueda funcionar, la tal como se muestra a continuación:

```
<script src="http://maps.google.com/maps/api/js?key=AIzaSyAkEifLsV_F43mT5ihpZmBkXRzEegZjVSE"
        type="text/javascript"></script>
```

Y a continuación se muestra como se declara el componente Gmap en el fichero xhtml.

```
<p:gmap id="gmap" center="#{mapaGestionFincaBean.centro}"
     zoom="#{mapaGestionFincaBean.zoom}" type="HYBRID"
     style="width:100%;height:500px" model="#{mapaGestionFincaBean.model}">

       <p:ajax event="stateChange" listener="#{mapaGestionFincaBean.onStateChange}" />
       <p:ajax event="pointSelect" listener="#{mapaGestionFincaBean.onPointSelect}"
               update="gmap, :formMenu" />
       <p:ajax event="markerDrag" listener="#{mapaGestionFincaBean.onMarkerDrag}"
               update="gmap, :formMenu" />
       <p:ajax event="overlaySelect" listener="#{mapaGestionFincaBean.onOverlaySelect}"
               update="eliminaMarkerWindow" />

</p:gmap>
```

En el anterior fragmento de código, se especifica que el componente este atento a los eventos: *stateChange*, *pointSelect*, *markerDrag* y *overlaySelect*.

- El evento *stateChange* se generan cuando el usuario cambia el zoom, mueve el mapa para centrar la vista en otro punto del mapa.
- El evento *pointSelect*, se genera cuando el usuario hace clic en un punto del mapa. Este evento contiene las coordenadas de ese punto, latitud y longitud.

- El evento *markerDrag*, se lanza cuando el usuario hace clic sobre una marca del mapa y la mueve. Este evento nos devuelve cual es la marca movida y su nueva coordenada.
- Y por último, el evento *overlaySelect*, se lanza cuando se hace clic sobre un polígono dibujado en el mapa.

De esta manera hacemos que el bean esté atento a los clics sobre el mapa, *onPointSelect*, que esté atento al mover una marca del mapa, *onMarkerDrag* y al hacer clic en una marca *overlaySelect*. Y se le indica que métodos del bean se tienen que ejecutar según el evento que se lance.

Por el contrario, el bean *MapaFincaBean*, que contiene otro componente Gmap, solo muestra información sobre las fincas en el mapa. Y en el que se permite realizar búsqueda sobre que fincas se ha cultivado que y las pinta sobre el mapa.

Y por último, se desarrolló el bean *PartePlantacionBean*, que permite visualizar los datos de los partes de plantación que se recibe, hechos a través de los dispositivos Android.

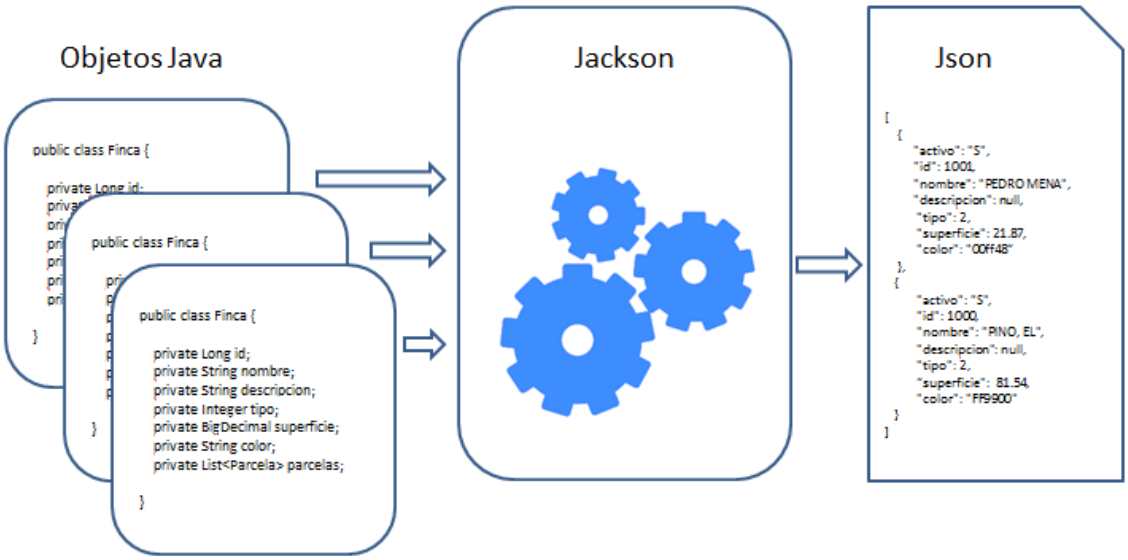
Una vez desarrollados los beans, se pasó a realizar los servicios Rest que la aplicación Android va a consumir. Estos servicios Rest tienen dos funcionalidades, la primera de ellas es enviar los datos necesarios a la aplicación Android para poder realizar un parte de plantación, como son los datos de las fincas, sus parcelas, el cultivo, etc. Y la otra función que es la de recibir los partes de plantación que se crean en la aplicación Android.

Un servicio Rest es un servicio el cual se accede mediante una url, este servicio puede necesitar para que se ejecute que se le envié algún dato o no. El paso de información puede ser a través de parámetros en la misma url o a través del cuerpo de la petición en formato JSON.
El formato JSON se ha hecho estándar para el intercambio de información a través de servicios Rest ya que ocupa menos que el formato XML, además definir de forma sencilla y clara la estructura de los datos y sus valores.

Ejemplo de una respuesta en JSON, de un listado de usuarios que pueden acceder a la aplicación, el usuario se define por el atributo id, activo, login, password y empleado.


```
[ {
            "id": 1000,
            "activo": "S",
            "login": "joscas",
            "password": "e40f01afbb1b9ae3dd6747ced5bca532",
            "empleado": null
  },
  {
            "id": 1001,
            "activo": "S",
            "login": "caselj",
            "password": "e40f01afbb1b9ae3dd6747ced5bca532",
            "empleado": null
  } ]
```

Para poder transformar un objeto Java en formato JSON y viceversa, se ha utilizado la librería *Jackson.* Esta librería automáticamente hace esta transformación. La elección de esta librería ha sido por ser una librería libre, sencilla de usar y con un muy buen rendimiento.




Para poder tener servicios Rest en una aplicación Java utilizando Java EE 7, debemos crear una clase que extienda de la clase javax.ws.rs.core.Application y añadir la anotación @ApplicationPath junto con el nombre de la uri que indica cual va a ser la uri para poder invocar los servicios Rest con una petición Http .

Esta es la clase del módulo web en la que se indica que todas las peticiones en la que la url contenga el texto "/rest", se debe buscar cual es el servicio Rest a ejecutar.

```
package com.gs.plantaciondroid.web;

import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

@ApplicationPath("/rest")
public class PlantacionDroidAplicacion extends Application {

}
```

Una vez hecha la configuración para disponer servicios Rest en el módulo web, pasamos a la creación de los servicios Rest.

Para desarrollar un servicio Rest, se debe desarrollar una clase Java, la cual hay que añadirle la anotación @Path. Para que se ejecute un método de este servicio, debemos indicar si la petición será de tipo GET O POST, con la anotaciones @GET o @POST, la anotación @Path para vincular la url con este método y si necesita un parámetro o no.

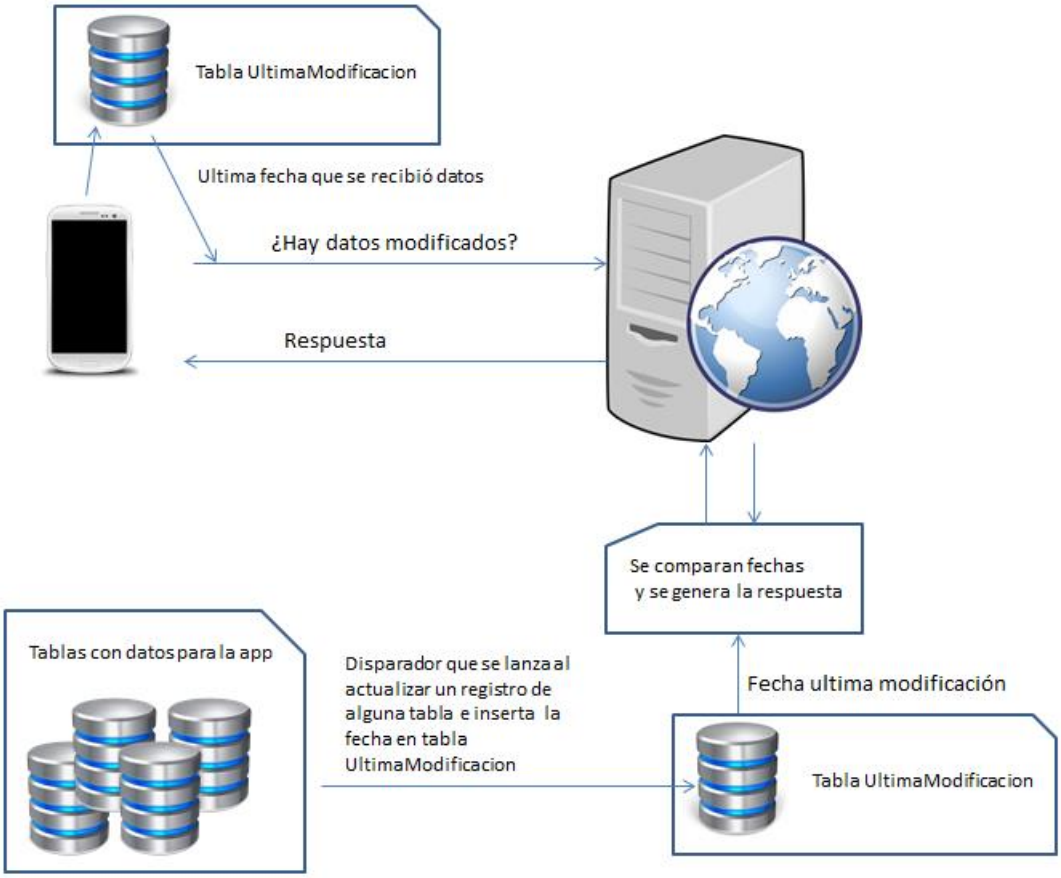
A continuación se muestra el código del servicio rest para obtener el listado de usuarios que tienen permiso para ejecutar una aplicación Android determinada. Para que se ejecute el método *permisoUsuario* la url debe ser la siguiente */PlantacionDroid/rest/usuario/WSUsuariosPermisoAplicacion*.

```
@Stateless
@Path("/usuario")
public class UsuarioRest {

    private static final Log LOGGER = LogFactory.getLog(UsuarioRest.class);

    @Inject
    private UsuarioServicio usuarioServicio;


    @GET
    @Path("/WSUsuariosPermisoAplicacion")
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    public Response permisoUsuario(@QueryParam("codigoAplicacion") String codigoAplicacion) {

        try {
            if(codigoAplicacion != null && !codigoAplicacion.equals("")) {

                List<Usuario> lista = usuarioServicio.buscarPorPermiso(codigoAplicacion);
                String response = JacksonJsonConverter.getInstance().toJson(lista);
                return Response.status(StatusRest.STATUS_OK).entity(response).build();
            }

        } catch(Exception e) {
            LOGGER.error("Error al obtener los usuarios con persmiso a la aplicacion: " +
                             codigoAplicacion, e);
            ResponseRest mensajeError = new ResponseRest(StatusRest.STATUS_ERROR_SERVIDOR,
                      "Error obtener lista de usuarios para: " + codigoAplicacion, e.getMessage());
            return Response.status(StatusRest.STATUS_ERROR_SERVIDOR).entity(mensajeError).build();
        }

        return Response.status(StatusRest.STATUS_ERROR_CLIENTE_RECURSO_NO_ENCONTRADO)
                .entity(new ResponseRest(StatusRest.STATUS_ERROR_CLIENTE_RECURSO_NO_ENCONTRADO,
                                "Datos incorrectos", ""))
                .build();
    }
}
```

Una vez visto como se hacen los servicios Rest, se desarrollaron los servicios Rest para dar soporte a la aplicación Android. Para ello se desarrollaron las siguientes clases Java que contienen los servicios Rest:

- UsuarioRest
- CampoRest
- PartePlantacionRest
- CoordenadaRest

Una vez desarrollado todos los servicios Rest, surgió la duda de como la aplicación Android podía actualizar los datos cuando estos se modifican, como por ejemplo si se da de alta una nueva finca, se elimina una variedad de un cultivo, etc.
Después de pensarlo durante un tiempo se vio que una posible solución para saber cuándo el modulo web le tiene que enviar datos actualizados a un dispositivo Android, es tener una tabla llamada *UltimaModificacion*, la cual contiene la fecha de la última actualización de cada una de las tablas con los datos que necesita la aplicación Android. Lo único que hace falta es que la aplicación Android debe consultar los servicios Rest pasándole la última fecha de cuando esta recibió datos actualizados. En el servicio Rest, se compara si esta fecha es anterior a la fecha que se tiene en la tabla *UltimaModificacion*, y si es así, es porque se han actualizado datos y se deben enviar estos datos. Así que en la aplicación Android se tiene otra tabla llamada igual, que contiene la fecha de la última recepción de datos.

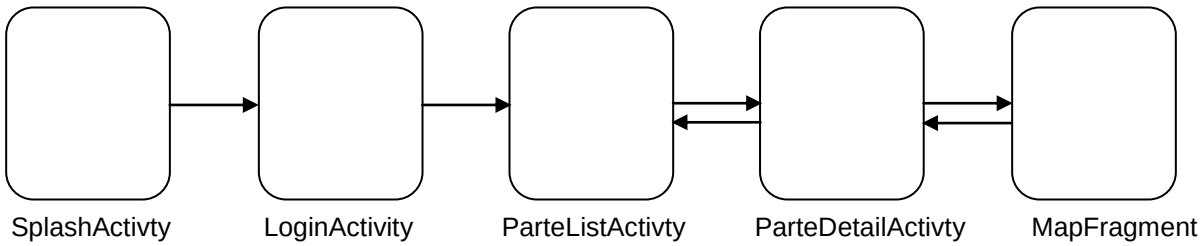



Y con esto se termina el desarrollo de la parte web y seguimos con la parte en Android.

El desarrollo de Android, se empezó desarrollando la estructura de la aplicación. Se crearon las siguientes ventanas:

- SplashActivity, ventana que contiene la ventana de bienvenida.
- LoginActovity, ventana para autenticar el usuario que quiere usar la aplicación.
- ParteListActivity, ventana que muestra un listado de los partes realizados por el usuario autenticado.
- ParteDetailActivity, ventana para el formulario de un parte de plantación en dispositivos pequeños o en tablets en posición vertical.
- ParteDetailFragment, ventana para mostrar el formulario de un parte de plantación en tablets en posición horizontal.
- MapFragment, ventana para mostrar el mapa donde se va a limitar la porción de tierra sembrada.

Y este sería el mapa de navegación entre estas ventanas:




Una vez montada la estructura de las ventanas junto con los componentes de la interfaz gráfica, se siguió con la implementación del modelo de clases. Tal como pasa en Java, en Android disponemos de varias librerías para poder mapear una clase con una tabla de una base de datos. En este caso se ha utilizado la librería SugarORM para tal propósito. Se ha escogido esta librería por su facilidad de uso y su potencia en comparación con otras librerías.
Para poder utilizar esta librería, solo debemos iniciar el contexto de la librería y que aquellas clases que quieran ser mapeadas en base de datos, que extiendan de la clase *SugarRecord*.

Después de desarrollar las clases del modelo de datos, se pasó a implementar las clases que accederían a los servicios Rest del módulo web. Para acceder o invocar un servicio Rest, lo que se debe hacer es establecer una comunicación http con la url del servicio Rest y pasarle los parámetros que necesita este para poder ejecutarse. Para ello tenemos varias formas de implementar esta funcionalidad, una de ellas es utilizar un objeto que establece conexiones Http e implementar toda la lógica de transformar los datos que se enviar, los datos que se reciben y además de implementar todo esto en hilos de ejecución en segundo plano con clases de tipo *TaskAsyn* para que la aplicación no deje al usuario sin poder trabajar con ella. O utilizar la librería Retrofit junto con la RxJava y RxAndroid que implementa todo eso, haciendo mucho más sencilla la

programación y centrándonos en que es lo que debemos hacer con los datos que se reciben. Estos datos pueden ser desde datos de tipo básico, numero, texto a una clase POJO o un listado de estos, siempre y cuando estos datos vengan en formato JSON.
Por tanto se optó por utilizar la librería Retrofit junto con RxJava y RxAndroid. A continuación se muestra la interfaz de un servicio para llamar a la url del servicio Rest que nos devuelve un listado de usuarios que pueden ejecutar la aplicación Android que tiene un código determinado:

```
public interface UsuarioServicio {

    String SERVICE_ENDPOINT = PlantacionDroidApp.URL_WEB_SERVER + "/usuario";

    @GET("/WSUsuariosPermisoAplicacion")
    Observable<List<Usuario>> getUser(@Query("codigoAplicacion") String app);
}
```

Y esta es su implementación donde se especifica que hacer con cada uno de los objetos de tipo *Usuario* que se reciben (método *onNext)*, que se tiene que hacer cuando se han recibido todos los objetos (método *onCompleted*) y que se tiene que hacer cuando se produce un error (método *onError*) :

```
UsuarioServicio service = ServiceFactory.createRetrofitService(UsuarioServicio.class,
UsuarioServicio.SERVICE_ENDPOINT);

service.getUser("plantaciondroid")
        .subscribeOn(Schedulers.newThread())
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(new Subscriber<List<Usuario>>() {
            @Override
            public final void onCompleted() {
                actualizaUltima(UltimaModificacion.TIPO_USUARIO);
            }

            @Override
            public final void onError(Throwable e) {

            }

            @Override
            public final void onNext(List<Usuario> lista) {
                for(Usuario usuario : lista) {
                    usuario.save();
                }
            }
        });
```

En este caso lo que se hace al recibir el listado de usuarios es guardarlos en la base de datos y cuando se termina de recibir datos entonces se ejecuta el método *actualizaUltima*.

Todas las clases que invocan servicios Rest para obtener los datos que necesita la aplicación, son ejecutadas desde la ventana de bienvenida, *SplashActivity*. Así de esta forma mientras el usuario se autentica, se comprueban si hay datos nuevos y se descargan y cuando el usuario acceda al formulario para crear un nuevo parte o modificar uno ya existente tendrá los datos actualizados.

Después de implementar esta funcionalidad, se implementó el código para autenticar que el usuario tiene permiso para acceder a la aplicación. Una vez que el usuario se autentica, es en este momento en el que se lanza el servicio para ver si tiene partes de plantación generados en otros dispositivos y si es así, se descargan para que los tengan accesibles. Para ello se ejecuta un servicio rest al que se le pasa la fecha del ultimo parte creado en el dispositivo, y es el servicio Rest quien comprueba si hay partes posteriores a esta fecha, y si es así, se envían al dispositivo.

La siguiente lógica que se desarrollo fue la de obtener el listado de partes de plantación del usuario, y que se muestren en el listado para tal efecto.

Después, se implementó la lógica para crear o modificar un parte de plantación. En la lógica de esta ventana se intenta obtener la posición del GPS, si se obtiene las coordenadas del usuario, se intenta buscar que esa coordenada este dentro de una finca. Si se identifica la finca, se intenta identificar en que parcela esta. En caso contrario de no poder identificar la finca, debe ser el usuario el que debe seleccionar la finca y la parcela en la que se ha plantado.

Y por último, se implementó la lógica para poder delimitar las coordenadas de lo plantado. Para centrar el mapa en la posición del usuario, se activa el GPS y una vez que se obtiene las coordenadas, se centra el mapa en esa posición.
El usuario, para delimitar la zona en la que se ha plantado, solo debe ir pulsando en el menú los vértices de esta zona.

Para solucionar el problema de que si se crea un parte de plantación, este no se puede enviar por falta de cobertura, se ha creado la clase *EnviarParte* la cual se guarda en la base de datos, en la que tiene el atributo del identificador el parte que no se ha podido enviar y la fecha de creación del registro. Una vez que se tiene registrado los partes que no se han podido enviar, se desarrolló un servicio el cual cada hora comprueba si el dispositivo tiene acceso a Internet y si es así entonces comprueba si hay partes por enviar. En el caso de que haya partes por enviar se envían y si todo es correcto se elimina el registro para que no se vuelva a enviar.

# 4. Conclusiones

La primera conclusión que he aprendido con este trabajo, es, que aunque se vaya a desarrollar una aplicación sencilla para un dispositivo móvil, este desarrollo no es trivial ya que se deben tener muchas cosas en consideración y más si existe una comunicación con un servidor.
Al pensar que el proyecto sería sencillo y no se tendría ningún problema para desarrollarlo, la planificación de las horas en las tareas de programación fue muy optimista. Este optimismo ha sido un error y ha hecho imposible poder implementar todas las funcionalidades que se habían especificado en los requisitos del proyecto. En la estimación de la duración de las horas de las tareas de programación se debería haber incrementado un porcentaje para estar cubiertos si surge algún problema en el desarrollo de alguna tarea como ha pasado a lo largo de este proyecto.

Así que una lección que he aprendido, es que, aunque un proyecto pueda parecer sencillo siempre hay que añadir al tiempo estimado un porcentaje más para posibles desviaciones por distintas causas y de esta manera poder llegar a tiempo para cumplir con los objetivos del proyecto.

Casi se ha logrado implementar todos los requisitos que se habían especificado en el proyecto. No se ha podido implementar uno de los principales requisitos del proyecto que era la delimitación automática de la siembra de un parte de plantación por falta de tiempo. Para suplir este requisito se optó por implementar la funcionalidad para que la delimitación de la siembra fuese manual.

Por otra parte, el modulo web no se corresponde exactamente con el prototipo del proyecto, aunque se ha intentado que se lo mas fiel posible. No se ha podido debido en parte al framework Prime Faces. Hay que decir que la delimitación de fincas es más sencilla de lo que se había planteado en el prototipo, simplificando esta tardea.

La futura línea del trabajo seria en mejorar el rendimiento para la actualización de los datos de la aplicación Android. Una posible mejora, sería que la aplicación Android en vez de comprobar si hay datos para actualizar cuando se accede a ella, se debería hacer que programáticamente aunque el usuario no entre a la aplicación se consulte si hay datos a descargar y descargárselos. Esta mejora se tendría que realizar ya que se puede dar el caso de que cuando se acceda a la aplicación no se tenga conexión internet y tenga que descargarse datos nuevos. Haciendo que la aplicación no tuviese los datos actualizados. Mientras, si se programa que cada cierto tiempo se comprueben si hay datos a descargar, y en el caso de que en ese momento no haya Internet dejar a la espera esta tarea a tener conexión a Internet, se incrementa en la fiabilidad en tener los datos actualizados.

Otra línea de trabajo seria mejorar la interfaz gráfica para la delimitación de las fincas en el módulo web y la gestión de estas. Haciendo que fuese más sencillo el seleccionar una parcela para delimitar y la consulta de fincas.

Para terminar, comentar que, aunque no se haya implementado una de las principales funcionalidad, estoy contento por el resultado. Ya que la funcionalidad alternativa implementada es más que necesaria. Esto es, porque si la aplicación no tiene Internet en el momento para delimitar el área de la siembra plantada en el campo, hace imposible obtener las coordenadas para guardarse junto al parte de plantación. Teniendo que hacer esta delimitación en un sitio donde se tenga cobertura, y por tanto haciendo imposible la delimitación automática. Así que esta funcionalidad alternativa que no estaba planteada en los requisitos del proyecto, durante el desarrollo de este, se ha visto que es una funcionalidad imprescindible para poder tener las siembras delimitadas.
Así que si se hubiese podido desarrollar la funcionalidad de la delimitación automática, se debería haber implementado la delimitación manual ya que se tiene que en algunas fincas no se tiene cobertura de datos para acceder a Internet. Y se tendrían partes sin delimitación de la plantación de ese parte.

# 5. Glosario

AJAX: acrónimo de Asynchronous JavaScript And XML (JavaScript asíncrono y XML), es una tecnología para desarrollar páginas web que sean interactivas.

API: es la abreviatura de Application Programming Interface su traducción es interfaz de programación de aplicaciones. Es un conjunto de funciones, procedimientos y subrutinas para ser utilizado por otro software.

Android Studio: es el entorno de desarrollo integrado oficial de la plataforma Android, para desarrollar aplicaciones Android.

Datasource: Es la traducción de fuente de datos y representa toda la configuración necesaria para configurar una conexión a una base de datos.

ERP: enterprise resource planning, son los sistemas de información gerenciales que manejan la producción, logística, distribución, inventario, envíos, facturas y contabilidad de la compañía de forma modular.

Eclipselink: es una librería que implementa la especificación JPA.

Framework: es un conjunto de librerías para facilitar el desarrollo de aplicaciones.

GPS: Sistema de Posicionamiento Global, es un sistema que permite determinar en toda la Tierra la posición de un objeto

Hectárea: Es una medida de superficie que equivale a 10.000 metros cuadrados.

Http: protocolo de transferencia de hipertexto, es el protocolo de comunicación que permite las transferencias de información a través de páginas web.

IDE: es la abreviatura de Integrated Development Environment y su traducción es entorno de desarrollo integrado. Es una aplicación para facilitar el desarrollo de software.

IP: es una dirección que identifica una interfaz de red de un dispositivo conectado a una red que utiliza el protocolo IP, Internet Protocol.

Jackson: es una librería Java para convertir objetos POJO en formato JSON y viceversa.

Java: es un lenguaje de programación de propósito general, concurrente, orientado a objetos tiene la peculiaridad de filosofía de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo

Java EE: Java Enterprise Edition es una parte de la plataforma Java que esta definida por varias API para desarrollar aplicaciones modulares y que se ejecutan en un servidor de aplicaciones.

JDK: Java Development Kit, es un software que provee herramientas de desarrollo para la creación de programas en Java.
JPA: Java Persistence API, define el estándar en Java que define una abstracción que nos permite realizar la integración entre POJOs y el sistema relacional de una base de datos.

JPQL: Java Persistence Query Language, es un lenguaje para hacer consultas para recuperar las entidades almacenadas en una base de datos relacional, además de poder modificar estas entidades y guardar estos cambios en la base de datos. Es parte de la especificación de JPA.

JSF: JavaServer Faces es una tecnología y framework para aplicaciones Java basadas en web. Simplifica el desarrollo de páginas web en aplicaciones Java EE y que usa JSP para su funcionamiento.

JSON: acrónimo de JavaScript Object Notation, es un formato de texto ligero para el intercambio de datos.

JSP: JavaServer Pages es una tecnología de la plataforma Java EE, que ayuda a los desarrolladores de software a crear páginas web dinámicas basadas en HTML y XML, entre otros tipos de documentos.

Marco de plantación: Es una medida la cual indica el número de plantas plantadas en una hectárea.

Maven: es una herramienta de software para la gestión y construcción de proyectos Java.

MVC: Modelo–vista–controlador (MVC) es un patrón de arquitectura de software, que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador.

Netbeans: es un entorno de desarrollo integrado libre, principalmente para el desarrollo de aplicaciones con el lenguaje de programación Java.

Observer: es un patrón de diseño que define una dependencia del tipo uno-a-muchos entre objetos, de manera que cuando uno de los objetos cambia su estado, notifica este cambio a todos los dependientes.

Patrón de diseño: Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Un patrón de diseño resulta ser una solución a un problema de diseño

POJO: son las iniciales de Plain Old Java Object. Hace referencia al desarrollo de clases simples y que no extiendes de ninguna clase de un framework. Normalmente estas clases solo tienen atributos para contener datos junto con sus métodos asociados para leer, métodos get, y escribir, métodos set, el valor para cada uno de estos atributos.

Prime Faces: es una biblioteca de componentes para JavaServer Faces (JSF) de código abierto que cuenta con un conjunto de componentes enriquecidos que facilitan la creación de las aplicaciones web.

REST: Transferencia de Estado Representacional interfaz entre sistemas que utilice directamente HTTP para obtener datos o indicar la ejecución de operaciones sobre los datos, en cualquier formato (XML, JSON, etc) sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes

Retrofit: es un cliente REST para Android y Java. Permite hacer peticiones http y descargar datos en formato JSON o XML de manera sencilla. Una vez descargado se transforman en un POJO para poder tratar los datos en la aplicación.

Rio: es un surco en línea recta, hecho en la tierra donde se planta el cultivo.

RxAndroid: es una librería Android que trata de facilitar la programación asíncrona y basada en eventos mediante el uso del patrón Observer.

RxJava: es una librería Java que trata de facilitar la programación asíncrona y basada en eventos mediante el uso del patrón Observer.

Script: es un archivo que contiene órdenes.

Servidor de aplicaciones: Un servidor de aplicaciones generalmente gestiona la mayor parte (o la totalidad) de las funciones de lógica de negocio y de acceso a los datos de la aplicación.

Servidor web: Contiene sitios web y utiliza el protocolo http para procesar las peticiones que los clientes le realizan.

SQL: son las siglas de Structured Query Language, lenguaje de consulta estructurada. Es el lenguaje utilizado para hacer consultas para recuperar datos en una base de datos relaciona, así como modificar estos datos.

SugarORM: es una librería para utilizar la persistencia de objetos en Android.

Tomcat: Apache Tomcat (también llamado Jakarta Tomcat o simplemente Tomcat) funciona como un contenedor de servlets y de JavaServer Pages. Se pueden desplegar aplicaciones web que estén desarrolladas con servlets y JavaServer Pages.

WildFly: es un servidor de aplicaciones Java EE, desarrollado totalmente en Java. Anteriormente se conocía como JBoss y desarrollado por JBoss Inc.

XML: siglas en inglés de eXtensible Markup Language, traducido como "Lenguaje de Marcado Extensible" o "Lenguaje de Marcas Extensible", es un meta-lenguaje que permite definir lenguajes de marcas desarrollado por World Wide Web Consortium (W3C). Y utilizado para almacenar datos en forma legible.

# 6. Bibliografía

GOOGLE DEVELOPERS. *Google Maps API*. <https://developers.google.com/maps/?hl=es-419> [consulta: 13 abril 2017]

PRIME FACES. *Gmap*. <https://www.primefaces.org/showcase/ui/data/gmap/basic.xhtml> [consulta: 16 abril 2017]

Tales of a Programmer. *Retrofit and RxJava, Android multi-threaded REST requests.* <http://randomdotnext.com/retrofit-rxjava> [consulta: 5 mayo 2017]

# 7. Anexos

## Instrucciones para la compilación de los proyectos

### Proyecto Web

Para compilar en el proyecto web, necesitamos tener instalado el JDK 8, y también la herramienta Maven. Si no disponemos del JDK 8 podemos descargarlo desde la página de Oracle. Y la página para descargar maven es https://maven.apache.org/. Es importante que a la hora de compilar el proyecto se tenga acceso a internet, ya que maven se descargara las librerías necesarias para la compilación del proyecto. Tambien es importante que se tenga en el Path, el ejecutable de maven para que este pueda ser invocado desde cualquier directorio.

Una vez que tengamos instalados el JDK y Maven, pasamos a compilar el proyecto. El primer proyecto que se tiene que compilar es el proyecto Modelo PlantacionDroid. Para compilarlo, lo que debemos hacer, es abrir una consola del sistema e ir al directorio de este. Una vez está estando en el directorio que contiene el fichero *pom.xml* del directorio, ejecutamos el siguiente comando:

```
> mvn clean package
```

Este sería el resultado si todo ha sido correcto.

```
caselj01@CASELJ01LAPTOP D:\master\tfm\src\modelo
> mvn clean package
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Modelo PlantacionDroid 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ modelo ---
[INFO] Deleting D:\master\tfm\src\modelo\target
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ modelo ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ modelo ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 41 source files to D:\master\tfm\src\modelo\target\classes
[WARNING] /D:/master/tfm/src/modelo/src/main/java/com/gs/modelo/base/ServicioDAOImpl.java: Some input files use unch
ecked or unsafe operations.
[WARNING] /D:/master/tfm/src/modelo/src/main/java/com/gs/modelo/base/ServicioDAOImpl.java: Recompile with -Xlint:unc
hecked for details.
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ modelo ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory D:\master\tfm\src\modelo\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ modelo ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ modelo ---
[INFO] No tests to run.
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ modelo ---
[INFO] Building jar: D:\master\tfm\src\modelo\target\modelo-1.0-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 4.537 s
[INFO] Finished at: 2017-06-01T20:19:27+02:00
[INFO] Final Memory: 34M/221M
[INFO] ------------------------------------------------------------------------
```

Esto generara el fichero *modelo-1.0-SNAPSHOT.jar* dentro de la carpeta *target*. En esta carpeta tenemos ahora que instalar este fichero en el repositorio local mediante el siguiente comando:

```
> mvn install:install-file -Dfile=modelo-1.0-SNAPSHOT.jar -DgroupId=com.gs -DartifactId=modelo -Dversion=1.0 -Dpackaging=jar
```

Y esta sería la salida, si se ha instalado correctamente:

```
caselj01@CASELJ01LAPTOP D:\master\tfm\src\modelo\target
> mvn install:install-file -Dfile=modelo-1.0-SNAPSHOT.jar -DgroupId=com.gs -DartifactId=modelo -Dversion=1.0 -Dpackaging=jar
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-install-plugin:2.4:install-file (default-cli) @ standalone-pom ---
[INFO] Installing D:\master\tfm\src\modelo\target\modelo-1.0-SNAPSHOT.jar to C:\Users\caselj01\.m2\repository\com\gs\modelo\1.0\modelo-1.0.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 0.726 s
[INFO] Finished at: 2017-06-01T20:24:08+02:00
[INFO] Final Memory: 7M/245M
[INFO] ------------------------------------------------------------------------

caselj01@CASELJ01LAPTOP D:\master\tfm\src\modelo\target
>
```

Ahora que ya tenemos el proyecto compilado e instalado, vamos a compilar y generar el modulo web para desplegarlo en un servidor de aplicaciones. Para compilar y generar el archivo, debemos irnos al directorio donde está el proyecto web PlantacionDroid y ejecutar el siguiente comando:

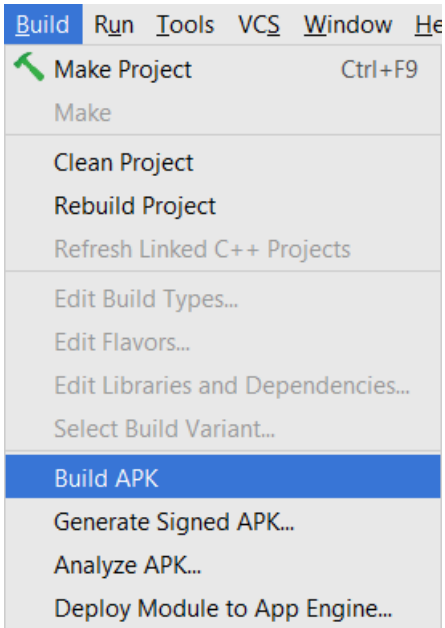
```
> mvn clean package
```

Si no se ha producido ningún error se habrá generado el fichero *PlantacionDroid.war* dentro del directorio target tal como se muestra en la siguiente imagen.

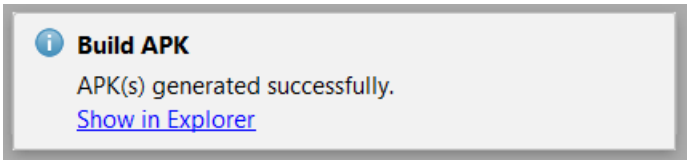
```
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building plantacionDroid 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ plantacionDroid ---
[INFO] Deleting D:\master\tfm\plantacionDroid\target
[INFO]
[INFO] --- maven-dependency-plugin:2.6:copy (default) @ plantacionDroid ---
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ plantacionDroid ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ plantacionDroid ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 18 source files to D:\master\tfm\plantacionDroid\target\classes
[WARNING] /D:/master/tfm/plantacionDroid/src/main/java/com/gs/plantaciondroid/web/BaseConverter.java: D:\master\tfm\plantacionDroid\src\main\java\com\gs\plantaciondro
id\web\BaseConverter.java uses unchecked or unsafe operations.
[WARNING] /D:/master/tfm/plantacionDroid/src/main/java/com/gs/plantaciondroid/web/BaseConverter.java: Recompile with -Xlint:unchecked for details.
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ plantacionDroid ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory D:\master\tfm\plantacionDroid\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ plantacionDroid ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ plantacionDroid ---
[INFO] No tests to run.
[INFO]
[INFO] --- maven-war-plugin:2.3:war (default-war) @ plantacionDroid ---
[INFO] Packaging webapp
[INFO] Assembling webapp [plantacionDroid] in [D:\master\tfm\plantacionDroid\target\plantacionDroid-1.0-SNAPSHOT]
[INFO] Processing war project
[INFO] Copying webapp resources [D:\master\tfm\plantacionDroid\src\main\webapp]
[INFO] Webapp assembled in [190 msecs]
[INFO] Building war: D:\master\tfm\plantacionDroid\target\plantacionDroid.war
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3.902 s
[INFO] Finished at: 2017-06-01T20:28:52+02:00
[INFO] Final Memory: 24M/235M
[INFO] ------------------------------------------------------------------------

caselj01@CASELJ01LAPTOP D:\master\tfm\plantacionDroid
```

## Proyecto Android

Para compilar el proyecto Android, lo que se debe hacer es abrir el entorno de programación Android Studio y abrir el proyecto PlantacionDroid que está dentro de la carpeta llamada Android. Si no se dispone del entorno de desarrollo, lo debemos descargar de la página oficial Android.
Una vez abierto el proyecto el entorno de desarrollo, hacemos clic sobre el menú *Build* y pulsamos sobre la opción de menú *Build APK*.




Si la app se genera correctamente, aparecerá la siguiente ventana



## Manual de usuario

### Modulo web

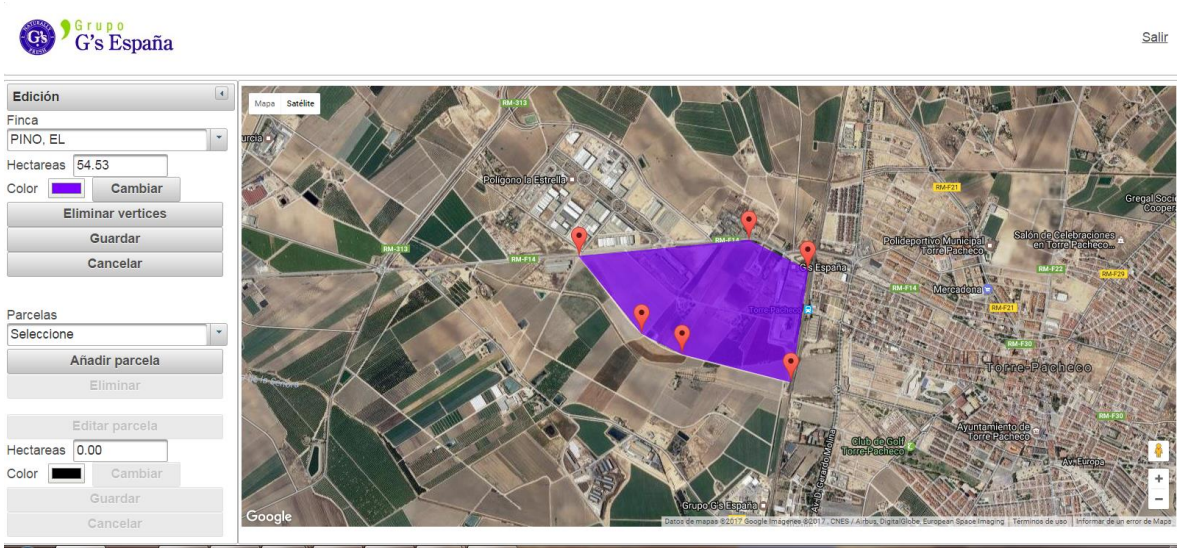
El usuario en el módulo web lo que puede hacer es, delimitar las fincas, delimitar las parcelas de estas y ver la información de los partes que se van haciendo.
Lo primero que tiene que hacer para acceder al módulo web, es autenticarse.

Una vez autenticado se redireccionara a una ventana principal con los enlaces que le está permitido acceder. Para este trabajo no se ha añadido ninguna política de permisos, con lo cual aparecen los tres enlaces de las ventanas desarrolladas.

- Gestión de Fincas
- Mapa de Fincas
- Consultar Partes Plantación

Si el usuario quiere delimitar una finca o delimitar alguna de sus parcelas debe hacer clic sobre *Gestión de Fincas*. Por el contrario si lo que quiere es ver las fincas en el mapa junto con información de esta, debe hacer clic sobre el enlace *Mapa de Fincas*. Y por último si lo único que se quiere, es revisar los partes de plantación, deberá hacer clic en el enlace de *Consultar Partes Plantación*.

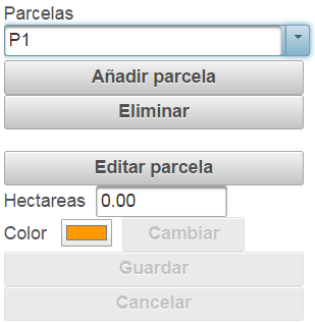
Esta es la ventana de gestión de fincas, con una finca seleccionada:




Como se puede apreciar en la parte izquierda tenemos la consola para poder gestionar las fincas y sus parcelas.
Para editar una finca debemos seleccionar la finca del selector de fincas y si esta está delimitada, entonces en el mapa aparecerá su superficie. Si se quiere añadir un vértice nuevo en la superficie de esta, solo debemos hacer clic sobre el mapa. Por el contrario si se quiere eliminar un vértice, solo debemos hacer clic sobre la marca de un vértice . Y por último si queremos cambiar de posición un vértice solo debemos hacer clic sobre uno de los vértices y sin soltar este lo movemos arrastrando hasta la posición correcta.
Podemos ver que hay un campo llamado *Hectáreas* el cual se va actualizando conforme se va modificando la superficie de la finca.
Para que las modificaciones hechas, se queden guardadas en la base de datos debemos pulsar sobre el botón de Guardar, y si queremos deshacer los cambios solo debemos pulsar sobre el botón Cancelar.

Para ir añadiendo parcelas a la finca seleccionada, debemos pulsar sobre el botón Añadir Parcela, que mostrara una ventana modal pidiendo el nombre de esta nueva parcela.

Para editar una parcela, debemos seleccionarla del selector de parcelas, tal como se muestra en la siguiente imagen:




Una vez seleccionada debemos pulsar sobre el botón de Editar parcela. Si pulsamos sobre este botón se deshabilitaran todos los botones menos los referentes a parcelas y se mostrara únicamente la delimitación de la parcela, tal como se ve en la siguiente imagen:

Si queremos guardar estos cambios, debemos pulsar sobre el botón de guardar, el cual volverá a activar los botones para trabajar con fincas y se volverán a mostrar tanto la superficie de la finca como el de resto de parcelas.

En todo momento podemos cambiar el color de una superficie, para ello solo debemos pulsar sobre el botón *Cambiar*, el cual mostrara una ventana para seleccionar el color que queremos.

El mapa de fincas es muy parecido a la ventana de Gestión de Fincas. Tiene en el lado izquierdo los criterios de búsquedas para buscar fincas. Y en el resto de la ventana, el mapa donde se mostraran las fincas que cumplan los criterios de búsqueda.
Los criterios de búsqueda son, por finca, por artículo cultivado, variedad del cultivo y por hectáreas cultivadas. Una vez especificado los criterios de búsqueda solo debemos pulsar el botón de *Filtrar.* En la siguiente imagen se ve la finca Pedro Mena, donde se dibuja la superficie de la finca, las distintas parcelas y los partes de plantación.

Podemos utilizar los botones de la parte izquierda inferior para mostrar una u otras superficies. En la siguiente imagen se muestra solo la superficie de la finca.

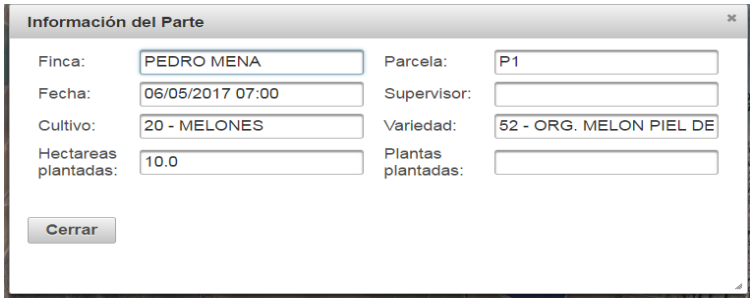
Si pulsamos sobre esta, aparecerá una ventana modal con información de la finca, como las hectáreas totales de la superficie, las parcelas que la conforman y los partes de plantación de esta finca.

En la siguiente imagen se muestra solo las distintas parcelas de la finca.

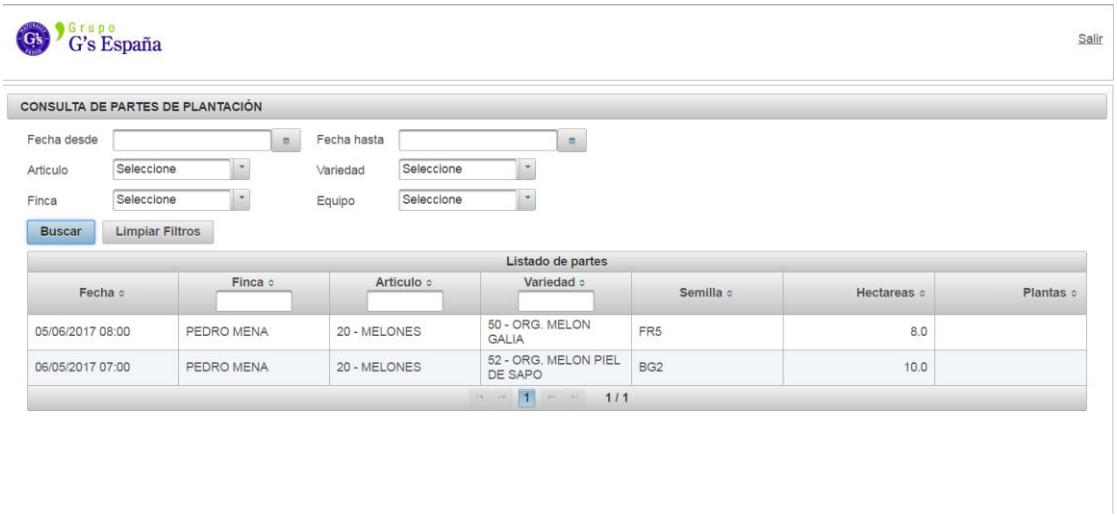
Y si clicamos sobre una parcela, nos mostrara información sobre esta. Aparecerá la siguiente ventana modal.

Y por último, así es como se vería si solo queremos mostrar los partes de plantación de esa finca.

Y si hacemos clic sobre un parte, se mostrara una ventana modal con los datos de este parte.



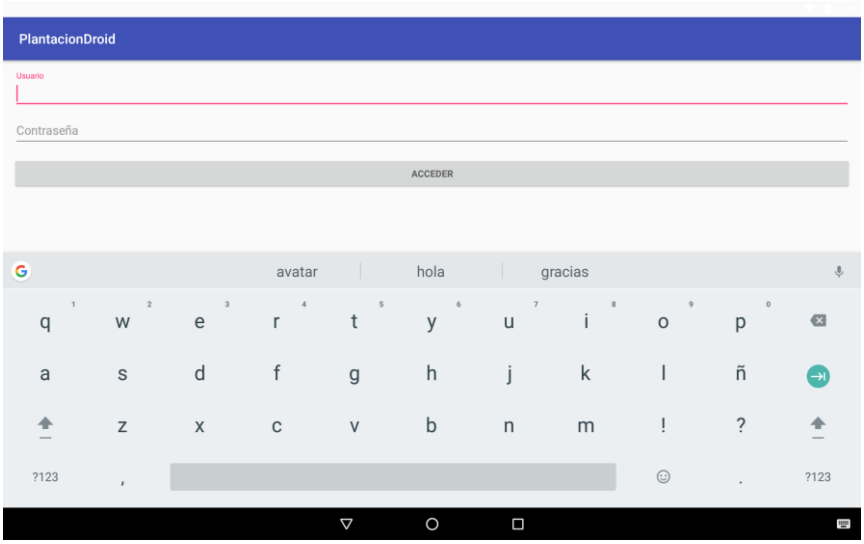

Y por último, no queda la ventana para consultar partes de plantación. Esta ventana tiene una serie de filtros para buscar los partes de plantación que coincidan con los criterios seleccionados. En la siguiente imagen se muestra la ventana donde se ha buscado todos los partes de plantación del sistema.



## Aplicación Android

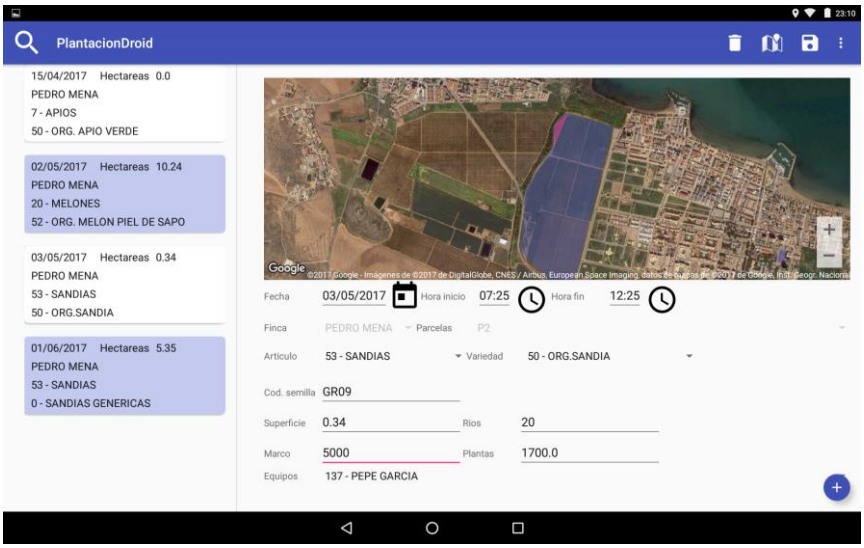
La primera vez que el usuario ejecute la aplicación esta debe tener cobertura a Internet para que se pueda descargar los datos para poder crear un parte de plantación.

El usuario para que pueda trabajar con la aplicación debe autenticarse usando su usuario y su contraseña. Podrá usar la aplicación siempre y cuando tenga permiso para ejecutar esta aplicación.




Si el usuario tiene permiso para ejecutar la aplicación accederá al listado de los partes de plantaciones hechos por él.

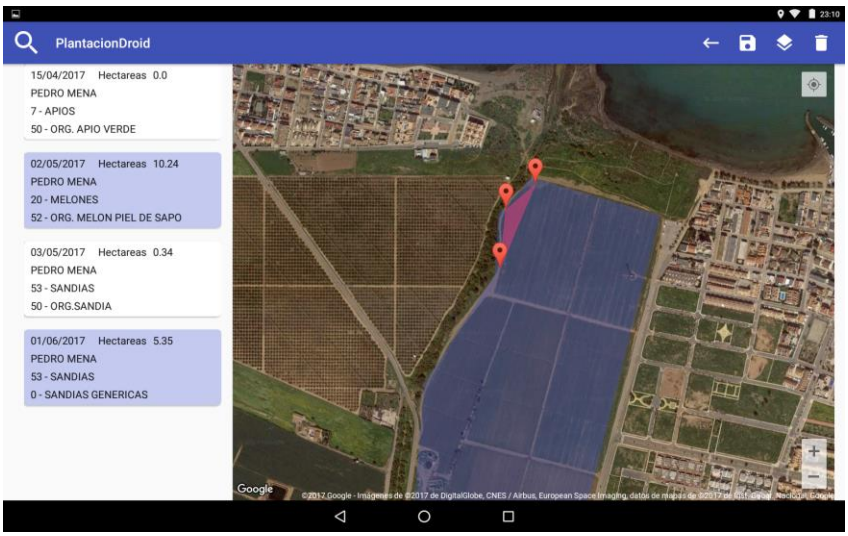
Para crear un nuevo parte se debe pulsar el botón , que está en la parte inferior derecha. Esto hace que se muestre un formulario en blanco para rellenarlo.
Por el contrario, si queremos editar en un parte de plantación ya creado, solo se tiene que pulsar sobre el parte de plantación del listado.




En el formulario de partes de plantación tenemos una serie de iconos con las distintas acciones que se pueden realizar, estos iconos se muestran en la parte superior derecha . Y estas son las acciones que tienen vinculadas:

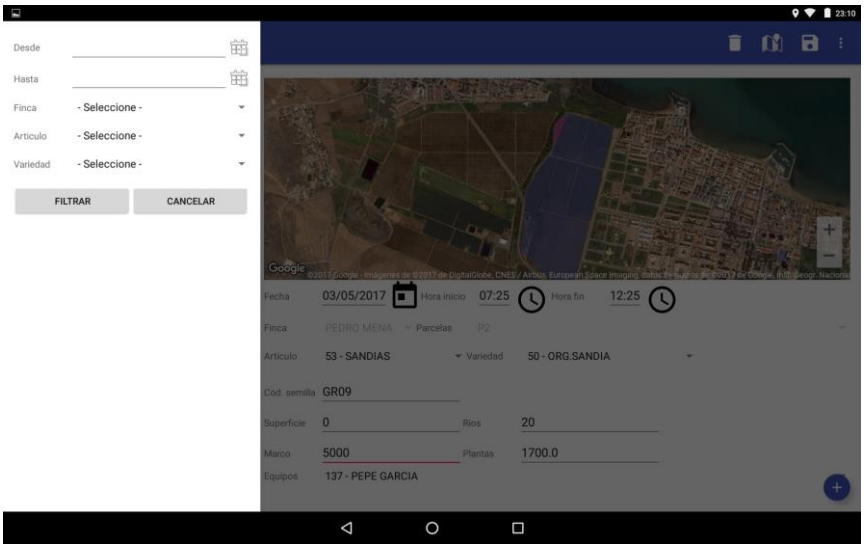
- elimina el parte que se está editando.
- guarda los cambios realizados del parte que se está editando.
- nos lleva a la ventana para la delimitación de la superficie plantada del parte de plantación.

Para delimitar en el mapa la siembra plantada en este parte de plantación tenemos que pulsar sobre el icono , esto hace que nos aparezca la siguiente ventana, en la que aparece un mapa para delimitar la siembra.



Para delimitar un parte de plantación, lo que debemos hacer es ir pulsando sobre este mapa. Con forme se pulse sobre el mapa irán apareciendo las siguientes marcas y se ira dibujando el área que delimita la plantación. También aparece en el mapa, el área coloreada de la finca que se ha indicado en el parte de plantación.
También podemos seleccionar una marca para moverla, en el caso de que no esté en la coordenada exacta, para ello debemos dejar pulsado sobre la marca un par de segundos y después arrastrar con el dedo la marca hasta la posición correcta.
Si queremos que no se muestre el área de la finca solo debemos pulsar sobre el icono y si queremos que se vuelva a mostrar solo tenemos que volver a pulsar sobre el mismo icono.

Por último, si queremos buscar un parte de plantación en particular, podemos filtrar el listado pulsando el icono para ello. Esto hace que se muestren los filtros por los que se puede buscar un parte. Una vez que se han seleccionado los criterios debemos pulsar sobre el botón *FILTRAR*. Si queremos deshacer el filtrado solo debemos pulsar sobre el botón *CANCELAR.*



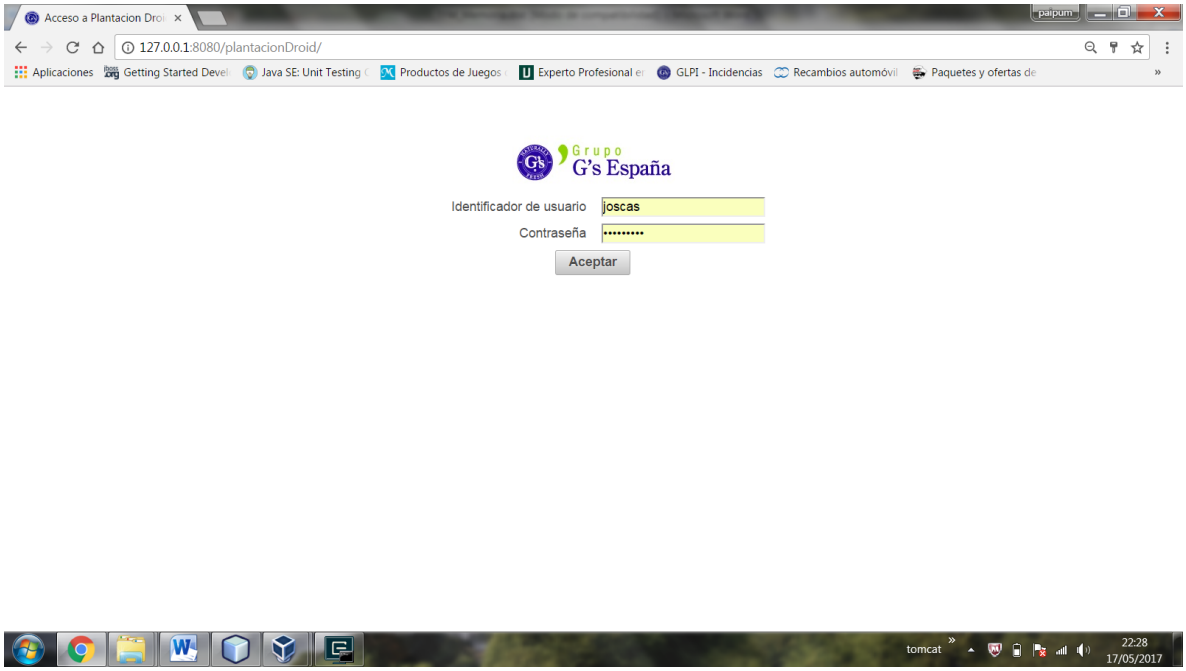
## Instrucciones para la instalación de las aplicaciones

Como la aplicación de Android por sí sola no puede funcionar porque necesita de un servidor que le proporcione los datos, empezaremos a instalar el modulo web.
Lo primero que debemos hacer, es crear las tablas que aparecen en el fichero *script.sql* en una base de datos Oracle, dentro de un esquema llamado MAESTRAS. Además en el script de base de datos se insertan en la base de datos, datos para empezar a trabajar con la aplicación Android.

Una vez que se tiene la base de datos con las tablas creadas y con datos, debemos configurar una conexión a esta base de datos en el servidor de aplicaciones que se quiera usar. Es muy importante en este punto, no confundir un servidor web con un servidor de aplicaciones ya que si se instala en un servidor web, como puede ser Tomcat, el modulo web no funcionara.

Se han incluido los driver para la conexión de la base de datos. Los driver están en el fichero llamado *ojdbc7.jar*. Según el servidor de aplicaciones que se use, este driver se tendrá que estar en un directorio u otro. Es importante que esta conexión, el datasource, tenga el siguiente nombre: *java:jboss/datasources/PlantacionDS*. Si el *datasource* no tiene este nombre, entonces el modulo web no podrá conectarse la base de datos. Una vez que se tiene configurado el *datasource*, lo siguiente que tenemos que hacer es desplegar el fichero *PlantacionDroid.war*.

Si todo es correcto podremos acceder al módulo web mediante un navegador web, poniendo la siguiente dirección: *http://127.0.0.1:8080/plantacionDroid/* y nos aparecerá la siguiente página:




Ahora que ya se tiene el modulo web funcionando, debemos instalar la aplicación Android en un dispositivo. Antes de instalar la aplicación mediante el fichero

*PlantacionDroid.apk*, debemos ver que IP tiene el servidor de aplicaciones. Cuando sepamos cual es la IP del servidor, debemos modificar el fichero *PlantacionDroidApp.java* del proyecto Android y modificaremos el valor del atributo *URL_WEB_SERVICE.*

Es necesario modificar el atributo del fichero *PlantacionDroidApp.java* porque si no, la aplicación Android no podrá conectarse al servidor para descargarse los datos que esta necesita .Este es el contenido del fichero:

```
public class PlantacionDroidApp extends Application {

    public static final String URL_WEB_SERVER =
"http://192.168.1.38:8080/plantacionDroid/rest";

    @Override
    public void onCreate() {
        super.onCreate();
        SugarContext.init(this);
    }

    @Override
    public void onTerminate() {
        super.onTerminate();
        SugarContext.terminate();
    }
}
```

Una vez modificado con la IP de nuestro servidor podemos generar la aplicación e instalarla en el dispositivo Android que queramos.