UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

# UNIVERSITAT POLITÈCNICA DE VALÈNCIA

## Escuela Técnica Superior de Ingeniería del Diseño

---

# APLICACIÓN MÓVIL PARA EL TELECONTROL DE EXPLOTACIONES CUNÍCOLAS.

*TRABAJO FINAL DEL*
**Grado de Ingeniería electrónica industrial y automática.**

*REALIZADO POR*
**Raquel Picazo Huerta**

*TUTORIZADO POR*
**Juan Carlos Martínez González**

**CURSO ACADÉMICO: 2019/2020**

# Agradecimientos

A todas las personas que me han apoyado a lo largo de este trabajo, muchas gracias por haber confiado en mí, incluso más que yo misma, y por darme la fuerza y motivación que necesitaba para conseguir lo que un día me propuse.

Gracias a mi tutor, sin el cual no habría sido posible la realización de este proyecto, por las horas de trabajo, por ayudarme a resolver las dificultades en el camino y por hacer que esta experiencia me sirva para aprender muchas cosas.

A mis padres y hermano, por su apoyo incondicional, por la paciencia que han tenido conmigo, por soportarme en los peores momentos y hacerme sentir bien en los buenos. Pero me gustaría agradecer en especial a mi padre, mi gran ejemplo a seguir, con el que he invertido mucho tiempo en este proyecto, gracias por aconsejarme y ayudarme a lo largo de esta experiencia, me ha encantado trabajar contigo.

Por último, agradecer a mi pareja por estar y permanecer siempre en los peores momentos durante toda la universidad. Gracias por saber calmarme y hacerme sentir bien.

# Resumen

A partir de una explotación cunícola la cual está automatizada, se pretende desarrollar la primera fase de una aplicación que facilitará el control de la actividad de la explotación a través del teléfono móvil. Este proyecto se realizará mediante la simulación del proceso de la explotación cunícola. La aplicación permitirá al cunicultor intercambiar información en tiempo real de manera fácil y sencilla, así como conocer el estado de la explotación de forma remota.

La aplicación informática servirá para recoger los datos de la granja diariamente con el fin de administrar y controlar tanto los parámetros ambientales de la explotación (temperatura, ventilación…) como los datos productivos de los animales existentes (alimentación…). A su vez, también permitirá controlar la trazabilidad del producto final a través del programa de gestión integral de datos.

Además, será posible crear gráficos e informes útiles mejorar el rendimiento y monitorizar el proceso de gestión de la granja y el funcionamiento de la climatización, para ayudar en el diagnóstico y control de temas productivos y sanitarios.

# Abstract

From a rabbit farm which is automated, it in intended to develop the first phase of an application that will facilitate the control of the activity of the farm through the mobile phone. This project will be carried out by simulating the process of the rabbit farm.The application will allow the rabbit farmer to exchange information in real time in an easy and simple way, as well as to know the status of the farm remotely.

The computer application will be used to collect data from the farm on a daily basis in order to manage and control the environmental parameters of the farm (temperature, ventilation...) and the production data of the existing animals (feeding...). In addition, it will allow to control the traceability of the final product through the integrated data management program.

Moreover, it will be possible to create useful graphs and reports to improve performance and monitor the farm management process and the operation of the climate control, to assist in the diagnosis and control of production and health issues.

# Resum

A partir d'una explotació cunícola la qual està automatitzada, es pretén desenrotllar una aplicació que facilitarà el control de l'activitat de l'explotació a través del telèfon mòbil. D'esta manera es permet al cuniculicultor intercanviar informació en temps real de manera fàcil i senzilla, així com conéixer l'estat de l'explotació de forma remota.

L'aplicació informàtica servirà per a arreplegar les dades de la granja diàriament a fi d'administrar i controlar tant els paràmetres ambientals de l'explotació (temperatura, ventilación...) com les dades productives dels animals existents (alimentació, rendiment, fertilidad...). I controlar la traçabilitat del producte final a través del programa de gestió integral de dades.

A més, serà possible crear gràfics i informes útils per a organitzar i millorar el rendiment, monitoritzar el procés de gestió de la granja i el funcionament de la climatització, per a ajudar en el diagnòstic i control de temes productius i sanitaris.

# Contenido

# Índice de figuras

# Índice de tablas

# 1. Introducción

El siglo XXI está siendo un siglo de evolución, el ser humano está desarrollando muchos avances tecnológicos con el fin de cumplir diferentes objetivos y funciones. El desarrollo de la tecnología y sus avances ha impulsado distintos campos de la industria, la investigación y educación entre otros. Por ejemplo, ya se puede percibir que la mejora es constante en casi todos los ámbitos que nos rodean: coches casi autónomos, smart TV's, Smartphones, hogares domotizados…
La revolución se ha adaptado a nuestras vidas para facilitar nuestra actividad, ha hecho de las cosas y procesos, algo más sencillo, rápido y útil, con avances como los siguientes: el Internet de las cosas (IoT), la inteligencia artificial, la comunicación M2M, el BigData, la fabricación digital e impresión 3D, etc.

Las tecnologías de la información y de la comunicación han venido haciéndose imprescindibles en la vida cotidiana de las personas en los últimos años. Hasta hace poco, las personas vivían la mayor parte de su vida ajenas a todas las facilidades que aportan estas tecnologías. El elemento más representativo de las nuevas tecnologías es sin duda Internet, ya que supone un gran avance para nosotros, éste ha cambiado y redefinido los modos de conocer y relacionarse del ser humano. La evolución de la tecnología avanza a pasos agigantados, ofreciendo cada vez nuevos servicios y mejoras.

Actualmente los smartphones están extraordinariamente extendidos en la sociedad, hasta tal punto que no sabríamos vivir sin ellos, y es que realmente cualquiera puede sacarles provecho, ya que nos hacen la vida más fácil.
Los Sistemas Operativos de los actuales smartphones (Android, iOS o Windows Phone) permiten desarrollar aplicaciones cada vez más útiles e interactivas. Se pueden encontrar aplicaciones de todo tipo, destinadas únicamente al entretenimiento, para ámbitos educativos y profesionales o también aplicaciones funcionales para las tareas del día a día como, por ejemplo, el acceso a la información, el almacenamiento de datos personales de forma segura o la compra de manera inmediata, sencilla y desde cualquier lugar.

Sin duda podemos sacar mucho partido del uso de las distintas aplicaciones, serán de importancia aquellas que sean capaces de ahorrar tiempo en la vida cotidiana, ayudando a la organización del usuario, así como aquellas ayuden a desempeñar labores profesionales de manera fácil con el fin de reducir el esfuerzo del personal y aumentar el rendimiento.

Los avances durante los últimos años han sido muchos, sin embargo, en el sector primario la evolución ha llegado de forma más tardía y causando una auténtica revolución. De este modo se ve necesaria la adaptación de las instalaciones de este sector a la revolución tecnológica que está viviendo nuestro siglo, para acercar la comodidad a trabajos pesados con el objetivo de facilitar y agilizar los procesos de producción y recolección y obtener el mayor rendimiento posible gracias a un mayor control.

Los avances que se están produciendo en este sector, se ven reflejados en notables mejoras tanto para el trabajador como para la explotación. Por lo que este acercamiento del sector a la tecnología puede atraer a nuevos trabajadores jóvenes a la agricultura y ganadería, ya que actualmente no tiene nada que ver con lo que se hacía antiguamente en cuanto al desempeño de tareas pesadas y la dificultad de éstas.

Debido a todo esto, porque es el presente y también el futuro, se pretende contribuir realizando una aplicación móvil objeto de este Trabajo Final de Grado.

## 1.1 Motivación

Como he comentado anteriormente, creo firmemente en la necesidad de acercar cada vez más las actividades del sector primario a la revolución tecnológica.
Personalmente me siento bastante ligada a este sector, ya que en mi familia se desempeñan numerosos trabajos relacionados a la ganadería y agricultura. Por esa razón y debido a mi interés en materias como la electrónica, la automática y la programación surge la motivación de la realización de este proyecto.

Este trabajo de fin de grado comenzó a raíz de un intento de ayudar a mi padre, el cual posee una explotación cunícola. Dicha explotación no se encuentra en nuestro lugar de residencia, por tanto, mi padre tiene que desplazarse unos 15 kilómetros todos los días para ir a trabajar.
Recuerdo muchas ocasiones en las que el hombre ha tenido que salir de casa a altas horas de la noche o incluso volver de sus vacaciones, para solucionar problemas que habían surgido en la explotación en su ausencia. Una aplicación móvil puede resolver este tipo de problemas ya que ofrece la posibilidad de obtener información de la explotación en tiempo real.

Además, con la aparición de la inseminación artificial en la cunicultura, surge la necesidad de controlar de forma más exhaustiva el ambiente de la explotación. Por lo que una aplicación de este tipo se ve cada vez más necesaria.

Al empezar el último curso de carrera y por las razones expuestas anteriormente me ha parecido una ocasión perfecta para desarrollar este tipo de aplicación móvil para mi Trabajo de Fin de Grado.

No solo me siento motivada por el desarrollo de una aplicación móvil (tecnología en auge y que me parece muy interesante), si no que a su vez ésta será muy útil en cuanto a rendimiento de la explotación y facilitará el trabajo diario de uno de mis seres más queridos evitando la necesidad de tener que desplazarse cuando ocurra algún inconveniente.

Otro de los aspectos que me animaron a llevar a cabo la idea de este trabajo, fue las ganas que tenía desde hace tiempo en comenzar el aprendizaje del lenguaje de programación Java. Antes del presente trabajo, no tenía conocimientos sobre él y por lo tanto esto supuso un reto para mí.

## 1.2. Historia de la cunicultura

La Cunicultura es el conjunto de técnicas y procesos llevados a cabo por el hombre para el aprovechamiento de la carne de conejo y sus productos. El concepto cunicultura procede del vocablo latino cuniculus (“conejo”) y de cultura (que puede asociarse al cultivo de algo).
La cunicultura, por lo tanto, consiste en la cría sistemática de estos animales. La carne de conejo forma parte del grupo conocido como carne blanca, se trata de una carne magra, con escasa cantidad de grasa.

La cunicultura industrial inició su desarrollo en España a finales de la década de los años 70 del siglo. Industrializar no significó explotar gran cantidad de conejos en una explotación, sino que fue un proceso de múltiples mejoras adaptables a cualquier dimensión de granja. La primera y muy importante, alojar a los animales en jaulas metálicas equipadas con comederos tolva y bebederos automáticos. La segunda, y no menos importante, alimentarlos con piensos comerciales. A partir de estas dos mejoras fundamentales, se han sucedido toda una serie de avances en lo que se refiere a los tres pilares de la explotación: mejora animal, formulación del alimento y planes sanitarios.
De los años 70 a los 80, la evolución fue muy importante y lo demuestran las estadísticas oficiales de aquellos tiempos que publicaban datos con incrementos constantes. De 25.200 toneladas de

carne de conejo en el año 1970, se publicó que a finales de los 80 España producía casi 100.000 toneladas de carne de conejo.
En la década de los años 80 el crecimiento, desarrollo y modernización de la cunicultura española fue espectacular. El concepto industrial se implantó gracias a las técnicas de la reposición y luego la sobreocupación.

Al inicio, las granjas destinaban unas jaulas para la maternidad (hembras y machos reproductores) y otras jaulas para el engorde. Las hembras siempre ocupaban su jaula que no desalojaban hasta ser eliminadas por muerte, grave enfermedad o reiterados fallos reproductivos. Eran sustituidas por hembras jóvenes captadas del engorde con una demora de hasta cuatro meses hasta que no producían.
Más adelante, se introdujeron en las granjas unas jaulas para la reposición donde se alojaban hembras jóvenes que sustituían a las eliminadas con lo que la demora productiva en una jaula era sólo de dos meses. La reposición supuso un avance productivo en las granjas empezando a primar aspectos de productividad gracias a mantener en granja conejas jóvenes (nulíparas) suficientes que, previamente cubiertas, empujaban la eliminación de las hembras presentes que no cumplían los parámetros productivos o manifestaban ciertas patologías.

También se ha observado la evidencia de la sobreocupación en las unidades cunícolas manteniendo en las granjas más hembras presentes (primíparas y multíparas) que Jaulas-Hembra instaladas (entendemos por JH la jaula que contiene un nidal). El trabajo se ha racionalizado gracias al manejo en bandas, organizando el trabajo de las operaciones en día fijo semanal, estimulando el celo de las reproductoras mediante la hormonación (PMSG) o el bioestímulo (lactación interrumpida) denominando a dicho manejo como ciclización.

Cuando aparece la inseminación artificial el sistema de funcionamiento cambió pasando a un manejo cíclico denominado “en bandas” y originado en Francia, basado en la sanidad gracias a unos módulos productivos en círculo cerrado donde entraban las reproductoras para parir, efectuar la lactación y desalojar el módulo dejando allí a sus gazapos hasta la venta para, una vez vendidos, proceder a un vacío sanitario. Este manejo supuso poder organizar el trabajo a día fijo semanal, optimizando tiempos horarios, y permitiendo nuevas estructuras productivas en las unidades de explotación.
De las cuatro bandas posibles: semanal, quincenal, trisemanal y única, esta última quiso ser la más implantada primada por razones eminentemente comerciales puesto que todas las conejas se inseminaban a la vez, las conejas que no quedaban o no parían eran eliminadas y todos los gazapos llegaban a la edad del sacrificio al mismo tiempo.

También resulta interesante observar cómo se ha ido desarrollando la cunicultura, más que evolucionando, respecto a los tipos de ambiente.
Iniciada la década de los años 80 del siglo XX, se divulga la bondad técnica (y económica) de las naves prefabricadas, tipo túnel, que empiezan siendo implantadas con Ambiente controlado en sobrepresión para derivar a instalaciones mixtas de Ambiente natural y controlado por depresión. Aparecen luego las naves prefabricadas con múltiples ofertas, todas en general, con ventilación natural y en algunas ocasiones asistida.
Llegados al presente siglo XXI, se preconizan las naves de ambiente controlado por depresión, con alimentación automática y limpieza de las deyecciones mecanizada.

Los animales que se explotan en cunicultura sí que han presentado una evolución espectacular. Al inicio de la industrialización de la cunicultura se disponía de razas puras y sus cruces. Más adelante empezaron a llegar híbridos procedentes Francia, con origen INRA, que crearon granjas de multiplicación en nuestro país y divulgaron una genética mejorada. Cuando en España, primero el IRTA y luego la UPV, se han dedicado esfuerzos en la mejora genética, tanto los híbridos españoles como los franceses están presentes en la mayoría de nuestras explotaciones.

Si bien la mejora de la prolificidad no ha conllevado a una destacada mejora de la productividad, la velocidad de crecimiento sí que ha conseguido mejorar la conversión del alimento.
Un alimento que ha pasado de único a doble (maternidad y engorde) para derivar a ofertas de alimentación triple (reproductoras, reproductoras con gazapos y gazapos solos) y llegar a situaciones de complicado control práctico. Alimento diferenciado para reproductores y para engorde, con alimentos medicados y piensos blancos finalizadores.

# 2. Objetivos y alcance

## 2.1. Alcance

Se podrá acceder a esta aplicación a través de un dispositivo móvil Android en el rango de versiones desde la 5.0 (Lollipop**)** a la 9.0 (Pie). Es decir, la aplicación está diseñada para garantizar la compatibilidad con versiones de Android API 21: 5.0 o superiores.

Lo que significa que un gran número de los dispositivos Android existentes serán capaces de hacer uso de todas las funcionalidades ofrecidas por la aplicación, en concreto, el 89,3 % de los usuarios.

## 2.2. Objetivos

El principal objetivo es el diseño e implementación de una aplicación para smartphones que permita gestionar una explotación cunícola concreta. La explotación cunícola para la que se realiza el presente trabajo consta de dos naves de conejos con controles independientes.
Se pretende aumentar el rendimiento y la producción de ésta, reducir los costes gracias al mejor aprovechamiento de los recursos, en resumen, una mayor eficiencia y comodidad para el ganadero o ganadera.

Uno de los objetivos que se pretende alcanzar con la ejecución del proyecto, es poner en práctica la mayor cantidad posible de conocimientos aprendidos durante el grado.
Estos se pueden resumir en los siguientes puntos:

- Realizar un análisis de los requerimientos que se nos propone.
- Definir y realizar un plan de trabajo.
- Aprender a utilizar nuevas plataformas de programación y uso de software.
- Diseñar la lógica del código y de base de datos, así como el análisis de los datos.
- Realizar la documentación y el diseño del trabajo final.

La aplicación dará soporte a las siguientes funcionalidades:

Control exhaustivo de los distintos parámetros ambientales de la instalación para conseguir un mayor bienestar de los animales al mantener la explotación en unas condiciones óptimas, con el fin de disminuir el número de defunciones y aumentar los kilos producción de carne, así como su calidad.
Partimos de que ya tenemos instalados una serie de sensores de los cuales obtendremos la información necesaria para activar o desactivar los diferentes actuadores que vamos a controlar. Sin embargo, como se ha comentado anteriormente, este proyecto se basa en el prototipo de la aplicación, por lo tanto, dejaremos la conexión de los sensores para futuras mejoras y se simulará la recogida de datos de los diferentes sensores a lo largo del proyecto.

También se pretende controlar la iluminación de las naves proporcionando las horas de luz necesarias y la alimentación de los animales ofreciendo a estos un racionamiento del pienso.

Otro de los objetivos perseguidos en este proyecto es llevar un control exacto de la cantidad de conejos existentes en todo momento en la explotación, es decir los kilos de peso vivo en la granja. Se pretende conocer la cantidad de kilos que se han producido en cada una de las bandas, así como utilizar esta información para un control eficiente de la ventilación de las diferentes naves. Esta información servirá además para facilitar el trabajo tanto al ganadero como al veterinario.

Cualquier incidencia que se produzca será resuelta de una manera más rápida y eficiente. Además, el programa informará al dueño o dueña de la explotación sobre su estado en determinados momentos que el mismo podrá establecer (situaciones de alarma), así como informar del estado del stock de los diferentes piensos, y a su vez mantener en la explotación unas condiciones óptimas para los animales de forma autónoma.

## 2.3. Herramientas

En este apartado se especifican las herramientas o recursos que han sido necesarios para el correcto desarrollo el proyecto.

### 2.3.1. Recursos humanos

Para la correcta realización del proyecto, y para garantizar que se cumplen los requisitos propuestos, han contribuido dos personas a lo largo de este.
Raquel Picazo Huerta ha desempeñado el papel de desarrolladora del proyecto y Juan Calos Martínez González ha ejercido un papel de ayuda y tutelaje.

### 2.3.2. Herramientas Software

- Office 365
  Se trata de una herramienta que nos permite crear, acceder y compartir documentos de Word, Excel, OneNote y PowerPoint y con la que puedes acceder a todos los programas en tiempo real. Además de estos programas, también tiene una serie de herramientas adicionales. Esta herramienta ha sido utilizada en varias ocasiones para la realización del proyecto, ya que nos ofrece diferentes programas:

  Como procesador de texto se ha utilizado Microsoft Word. Este software nos ha permitido la redacción de documentos, así como la modificación del formato de estos para su mejor visualización a lo largo del desarrollo.

  Otra de las herramientas utilizadas ha sido Microsoft Teams, que es un espacio de trabajo basado en chat de Office 365 diseñado para mejorar la comunicación y colaboración de los equipos de trabajo de las empresas, reforzando las funciones colaborativas de la plataforma en la nube. Se ha empleado para la comunicación con el tutor del proyecto, para las revisiones de este y la resolución de las dudas que han podido surgir.

  También se ha utilizado Power Point, que es uno de los programas más populares de Microsoft. Se trata de un software que permite realizar presentaciones a través de diapositivas. El programa contempla la posibilidad de utilizar texto, imágenes, música y animaciones. Ha sido utilizado para la realización de la presentación del proyecto al tribunal.

- Android Studio (Versión 3.6.1,2020)
  La principal herramienta en el desarrollo del proyecto ha sido Android Studio.
  Android Studio es el entorno de desarrollo integrado (IDE) oficial de Google para el desarrollo de aplicaciones para Android, basado en IntelliJ IDEA.
  Incluye Plugins como ADT (*Android Development Tools*) o AVD (*Android Virtual Device*) para poder desarrollar la aplicación y realizar pruebas sin necesidad del dispositivo físico, y Android SDK, el kit de desarrollo que necesitamos descargar para las aplicaciones Android, así como las librerías necesarias para el desarrollo en sus distintas versiones.

- SQLite (Versión 3.32.3, 2020)
  En cuanto al almacenamiento de datos, se utiliza SQLite ampliamente utilizado como bases de datos para Android. SQLite es una librería que implementa un motor de base de datos SQL transaccional. Esta librería es gratuita, de código abierto y además está integrada en el IDE de Google, y por lo tanto no ha sido necesario ninguna descarga ni instalación adicional.

### 2.3.3. Herramientas Hardware

A continuación, especificaremos las herramientas hardware que se han necesitado en la realización del proyecto, indicando también sus características.

- Pc Portatil Medion
  - Procesador: Intel(R) Core (TM) i7-7500U CPU @ 2.70GHz 2.90GHz.
  - Memoria RAM: 8 GB.
  - Gráficos (GPU): Intel HD Graphics.
  - Sistema Operativo: Windows 10, procesador 64 bits
- Teléfono móvil Xiaomi Mi 9T
  - Número del modelo: M1903F10G
  - Versión de Android: 9 PKQ1.190302.001
  - Capacidad total del dispositivo: 64GB
  - Memoria RAM: 6 GB
  - Procesador: Octa-core Max 2.2GHz
  - Tamaño: 2340x1080 píxeles, 6.39 pulgadas
  - Batería: Li-Ion 4000mAh

# 3. Estudio de necesidades, factores a considerar: limitaciones y condicionantes.

En este apartado se muestran los requisitos que se han tenido en cuenta a la hora del desarrollo de la aplicación. En primer lugar, se exponen los factores generales que se deben tener en cuenta y a continuación se mostrarán los requisitos para las diferentes ventanas a desarrollar.

A la hora de visualizar en la aplicación móvil el estado de la explotación, es importante que se refleje la información más relevante de manera resumida e intuitiva. Por ello será útil una pantalla general donde se muestre principalmente el control ambiental y desde donde se pueda acceder a cada una de las ventanas que controlan los distintos factores.

También sería interesante almacenar los datos ambientales y productivos para su posterior estudio, así como representarlos gráficamente para facilitar su comprensión y comparación con el fin de controlar la regularidad de la explotación.

## 3.1. Iluminación

Actualmente las naves cunícolas industriales carecen de iluminación natural, por lo que hay que proporcionarles una iluminación artificial, dado que las reproductoras relacionan las horas de luz con el ciclo reproductivo.

El programa de iluminación deberá controlar las horas de luz que necesitan los animales, encendiendo y apagando el sistema de iluminación cuando sea necesario.

Para ello el usuario tendrá que determinar el intervalo de duración en el que la luz deba estar encendida y la hora a la cual se activará el sistema de iluminación.

La iluminación consta de dos fases, normalmente las horas de luz al día oscilan entre 10-12 horas, exceptuando una semana antes de la inseminación en la cual se deberá aplicar un “flushing” (estímulo lumínico) aumentando las horas de luz a 16 horas durante dicha semana. Con este estímulo se consigue mejorar el celo de las reproductoras de cara a la inseminación.

## 3.2. Ventilación

Se requiere que el programa de ventilación se base en una ventilación de carga, lo que quiere decir que la nave se ventilará en función de los kilos vivos de animales que se encuentran en ella.

Partimos de que sabemos cuánto pesa cada conejo dependiendo del día del ciclo en el que se encuentre la nave. Es decir, se debe programar previamente cuanto pesa cada conejo para cada día del ciclo, cuanto va a ir creciendo a medida que pasa el tiempo.

El programa calculará cuantos kilos hay, para ello el usuario debe introducir cuantas madres y cuantas camadas hay en la explotación, así como el número de gazapos que hay en cada camada.

## 3.3. Calefacción

El sistema de calefacción consta de dos cañones de propano cuya función es mantener una temperatura de confort dentro de la nave. Estos cañones están situados orientados frontalmente a la entrada de aire de la nave, colocados en la parte superior. La calefacción estará activada todo el tiempo necesario hasta que la temperatura de la nave alcance la temperatura necesaria.

## 3.4. Refrigeración

La refrigeración es evaporativa, es decir, se lleva a cabo gracias a un cooling. Un cooling es un panel humidificador que consigue que el aire se enfríe cuando éste pasa a través de él aportándole humedad.
Por lo tanto, el sistema de refrigeración tiene como objetivo mantener la temperatura de confort dentro de la nave cuando en el exterior hace calor.

## 3.5. Alimentación

En cuanto al sistema de alimentación se controlará el encendido y el apagado de los silos. La aplicación deberá calcular a qué hora se debe dejar de alimentar a los conejos a partir de la información que el usuario introduce en el programa, esta será, la hora de inicio del racionamiento y la duración del mismo.

Además, es necesario conocer cuando los silos se están quedando sin pienso, concretamente, se emitirá una señal de alarma cuando en cualquiera de los silos disminuya su pienso por debajo del 20 por ciento.

## 3.6. Datos técnicos

En cuanto a los datos técnicos de los animales de la explotación, se pretende llevar un control de cada una de las bandas existentes en esta. Será necesario guardar en una base de datos las bandas creadas y su fecha, así como la información más relevante. Con todos estos datos que el usuario irá introduciendo a lo largo de cada ciclo, obtendremos una gran variedad de información muy útil para mejorar el rendimiento de la explotación.

# 4. Planteamiento de soluciones alternativas y justificación de la solución adoptada

En este apartado se hará un análisis de las distintas tecnologías disponibles para desarrollar la aplicación móvil y se justificará por qué se ha seleccionado una u otra.
Se han estudiado los principales tipos de aplicaciones móviles, los sistemas operativos móviles más utilizados y por ultimo los diferentes entornos de desarrollo para Android.

## 4.1. Tipos de aplicaciones móviles

Antes de comenzar el desarrollo de una aplicación es importante elegir el tipo de aplicación que queremos desarrollar. Hay tres tipos de aplicaciones móviles: aplicaciones nativas, aplicaciones web y aplicaciones híbridas. En este apartado se explicará cada una de ellas.

### 4.1.1. Aplicaciones nativas

Una aplicación nativa es la que se desarrolla de forma específica para un determinado sistema operativo, es decir, se desarrolla en el lenguaje específico de la plataforma original. En este sentido, si desarrollamos para Android lo haremos con C++, Java y XML o con Kotlin y sin embargo si lo hacemos para dispositivos con iOS lo haremos con Objective-C y Swift.

Sus ventajas son las siguientes:

- Una de sus principales ventajas es la velocidad de ejecución que permite obtener el mejor rendimiento posible que con cualquier otro tipo de app.
- Posibilidad de tener acceso a todas las funciones del terminal y dotar de esta forma de mayor funcionalidad a la aplicación
- Mejor experiencia de usuario.
- Mayor escalabilidad de la app en términos de tamaño, que no afectará al funcionamiento de la misma.
- Mayor visibilidad en la App Stores.

Los inconvenientes que presentan este tipo de aplicaciones son:

- Generalmente son más caras de desarrollar ya que, si queremos publicar la app para Android y para iOS, el desarrollo será diferente para cada plataforma.
- Posiblemente no podamos tener dos aplicaciones idénticas para las dos plataformas, pues cada una de ellas tiene sus funciones y por lo tanto los desarrollos no podrán ser idénticos.

### 4.1.2. Aplicaciones webs

Las aplicaciones web se desarrollan en lenguajes muy conocidos, por ejemplo, *HTML, Javascript* y *CSS*. Estas aplicaciones web se ejecutan directamente dentro del propio navegador web del dispositivo a través de una URL, lo que por una parte es una ventaja al no ser necesario instalarlas, pero por otra es un inconveniente ya que no pueden estar visibles en las tiendas virtuales de distribución de aplicaciones.

Entre sus ventajas destacan las siguientes:

- Proceso de desarrollo más sencillo y económico, ya que el código base se puede utilizar en múltiples plataformas.
- Al ser una URL a una web, el usuario siempre dispone de la última versión de la aplicación.
- El contenido se adapta a la pantalla adquiriendo un aspecto de navegación APP.

Los inconvenientes de las aplicaciones web son:

- Acceso muy limitado a las características hardware del dispositivo.
- Requiere conexión a internet para su utilización además de un gasto de datos que no siempre es posible por parte del usuario.

- No ofrecen una experiencia de navegación tan buena al usuario, el rendimiento empeora ya que no pueden extraer todo el potencial del dispositivo al no tener acceso a totalidad de las API´s y a los recursos.

### 4.1.3. Aplicaciones híbridas

Estas aplicaciones son una combinación de las dos anteriores, empleando las ventajas de cada una de ellas. Este tipo de aplicaciones se desarrollan con lenguajes propios de las aplicaciones web, es decir, HTML, JavaScript y CSS.

Las principales ventajas de estas aplicaciones son:

- Se adaptan al tamaño de pantalla del dispositivo y se ejecutan sobre el navegador nativo del sistema operativo sobre el que se ejecutan.
- Con un solo y único desarrollo, pueden ser utilizadas en las diferentes plataformas.
- Posibilidad de acceder a la mayor parte del hardware del dispositivo.

Este tipo de aplicaciones, aunque menos que las anteriores, también tiene algunos inconvenientes:

- Rendimiento de este tipo de aplicaciones es menor que el de una aplicación nativa.
- Experiencia de usuario más propia de una aplicación web que de una app nativa.
- A nivel estético, tampoco son tan atractivas como las nativas debido a las limitaciones de la tecnología de desarrollo utilizada.

### 4.1.4. Elección tipo de aplicación

Valorando las ventajas y los inconvenientes de los distintos tipos de aplicaciones mostrados en los apartados anteriores y teniendo en cuenta los objetivos del proyecto, finalmente, la opción elegida será una aplicación nativa.

Con las aplicaciones nativas obtenemos el mejor rendimiento posible y tenemos acceso a cualquier elemento del dispositivo. Es la técnica más cara, pero también es la que proporciona mejores resultados. En el caso de este proyecto se busca que el programa responda con cierta velocidad en cuanto a la consulta de datos y funciones del dispositivo, y que la navegación por la pantalla sea fluida y rápida.

Otro de los objetivos de este proyecto es desarrollar una aplicación móvil que ofrezca una buena experiencia al usuario, las aplicaciones nativas generalmente tienen la mejor experiencia de usuario al mostrar un diseño visual más acorde al sistema operativo en el que se encuentra la aplicación. La interfaz debe ser intuitiva, sencilla y agradable.
En este caso la aplicación se desarrollará para un único sistema operativo con lo que no se tendrán que crear varios códigos, y el coste del desarrollo será más reducido.

## 4.2. Sistemas operativos móviles

El Sistema Operativo es un programa principal o un conjunto de programas que gestionan los recursos hardware del dispositivo móvil y que provee los servicios a las aplicaciones de usuario. Es decir, son órdenes que controlan los procesos de funcionamiento básico de cualquier dispositivo o equipo informático.
Hay muchos sistemas operativos en el mercado y cada vez hay más, ya que en este sector nacen constantemente nuevas empresas para innovar con su tecnología en el mercado. No obstante, hay tres sistemas operativos que son los que marcan la diferencia y que mejores prestaciones ofrecen a los usuarios.

### 4.2.1. Android

Android es un sistema operativo basado en el núcleo de Linux. Fue diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes o tablets; y también para relojes inteligentes, televisores y automóviles. Inicialmente fue desarrollado por Android Inc., empresa que en 2005 fue comprada por Google. Ese mismo año empiezan a trabajar en la creación de una máquina virtual Java optimizada para móviles (Dalvik VM). Aunque no fue hasta 2008 cuando se popularizó, gracias a la unión al proyecto de Open Handset Alliance, un consorcio formado por 48 empresas de desarrollo de hardware, software y telecomunicaciones, que decidieron promocionar el software libre, pero ha sido Google quien ha publicado la mayor parte del código fuente del sistema operativo.
Tiene una gran comunidad de desarrolladores que crean aplicaciones para extender la funcionalidad de los dispositivos. Se ha superado el millón de aplicaciones disponibles para la tienda de aplicaciones oficial de Android.
Durante el año 2010, Android se consolida como uno de los sistemas operativos para móviles más utilizados, con resultados cercanos a iOS e incluso superando al sistema de Apple en EE.UU. En 2012, Google cambia su estrategia en su tienda de descargas online, reemplazando Android Market por Google Play Store, donde en un solo portal unifica tanto la descarga de aplicaciones como la de contenidos.

Uno de los aspectos fundamentales del sistema operativo de Android fue su orientación a la multiplataforma, algo realmente novedoso, debido a que hace unos años, un sistema operativo se asociaba a un único dispositivo. Rápidamente esta característica hizo que Android alcanzara sus objetivos, convirtiéndose en el sistema operativo más utilizado.

Las principales ventajas de este sistema operativo son:

- El código de Android es abierto: Google liberó Android bajo licencia Apache.
  Existe una gran cantidad de aplicaciones disponibles, en su mayoría de uso gratuito.
  Además, la libertad de código permite adaptar Android a bastantes otros dispositivos además de teléfonos móviles. Está implantado en tablets, GPS, relojes, microondas, etc.
- Sistema multitarea: el sistema Android es capaz de hacer funcionar a la vez varias aplicaciones y además se encarga de gestionarlas, dejarlas en modo suspensión si no se utilizan e incluso cerrarlas si llevan un periodo determinado de inactividad.
- Es muy personalizable, permite modificar la apariencia del sistema operativo y crear una interfaz intuitiva a gusto del consumidor ya que los múltiplos widgets contribuyen a ampliar la comodidad del usuario.
- Libertad para instalar aplicaciones. Permite instalar aplicaciones, aunque sean de origen desconocido, y no las restringe tanto como otros sistemas operativos.
- Gama más amplia de precios para escoger.
- Cuenta con la mayor comunidad del mundo siempre en constante movimiento.

Algunos inconvenientes de Android son:

- A pesar de ser una ventaja el ser un sistema multitarea, el hecho de tener varias aplicaciones abiertas hacen que el consumo de la batería aumente, lo que se puede solucionar con algunas aplicaciones para optimizarla.
- Android se encuentra muy fragmentado, lo que provoca problemas de incompatibilidad con algunas aplicaciones del Market que funcionan en determinadas versiones de Android.
- La libertad para instalar cualquier tipo de aplicación puede derivar en problemas de seguridad en el dispositivo.

### 4.2.2. iOS

iOS es un sistema operativo móvil lanzado y utilizado por la multinacional Apple Inc. Su nombre proviene de iPhone OS. Es decir, iPhone Operative System o Sistema Operativo de iPhone. Fue desarrollado originalmente para el iPhone (teléfono de la marca Apple), aunque después se ha usado en otros dispositivos de la compañía como reproductores de música iPod touch y tabletas iPad. Tiene la segunda mayor base de smartphones instalada en todo el mundo después de Android. Este sistema operativo no permite trabajar utilizando Adobe Flash ni Java lo que supone que sea poco compatible el desarrollo en paralelo de aplicaciones Android e iOS, ya que éste último es de código cerrado. Se encuentra únicamente en los dispositivos Apple, ya que son los únicos que lo pueden implementar y esto limita de manera notable su cuota de mercado.
Las principales ventajas de iOS son:

- Calidad de las aplicaciones, ya que antes de estar disponibles en App Store, pasan por números filtros manuales.
- Posee una elevada seguridad frente a las amenazas cibernéticas.
- Buena optimización de los recursos de los terminales que permite una mayor duración de la batería.
- Permite la sincronización de todos los dispositivos Apple de forma fácil y sencilla.
- Interfaz intuitiva.
- Cuenta con un asistente virtual (Siri)

Sin embargo, iOS también cuenta con una serie de desventajas:

- Precio elevado de sus terminales. Esta es una de las mayores desventajas de los terminales iOS. Muchos otros terminales que tienen las prestaciones similares ofrecen precios mucho más competitivos que Apple. Además, es necesario uno de estos terminales para comprobar el funcionamiento de la aplicación a desarrollar.
- El precio para poder desarrollar aplicaciones iOS también es elevado, ya que los desarrolladores tienen que abonar una cuota de 99$ anuales para poder acceder a las herramientas de desarrollo, y para poder subir las aplicaciones a la tienda.
- El único entorno de desarrollo disponible para este lenguaje de programación, Xcode, es propiedad de Apple. Para poder instalar este IDE es necesario tener un ordenador con el sistema operativo Mac OS X y una cuenta de Apple.
- El lenguaje de programación empleado es poco intuitivo y difícil de usar.
- Carece de la libertad de tener un sistema operativo de código abierto, limitando a los usuarios a utilizar únicamente las aplicaciones de la App Store.
- No está permitido modificar la API de cualquier componente del framework, restando así libertad a los desarrolladores.

### 4.2.3. Windows 10 Mobile

Windows 10 Mobile (anteriormente llamado Windows Phone) es un sistema operativo móvil desarrollado por Microsoft y diseñado para teléfonos inteligentes y tabletas. Es parte de las ediciones de Windows 10 y sucesor de Windows Phone 8.1.
El objetivo principal de este sistema operativo móvil es llevar la integración y unificación con su homólogo de PC Windows 10, y proporcionar una plataforma para los teléfonos inteligentes y pequeñas tabletas de 8 pulgadas de tamaño. Está diseñado para ser similar a las versiones de escritorio de Windows estéticamente.

Es interesante de este sistema operativo móvil la funcionalidad Continuum, que permite convertir el teléfono en un PC portátil con un simple accesorio dedicado para conectarse a un monitor o TV y conectar un ratón inalámbrico o un teclado.

### 4.2.4. Elección de sistema operativo

Como se ha mencionado anteriormente la aplicación que se va a desarrollar va a ser de naturaleza nativa, por lo que habrá que desarrollar un código u otro según el sistema operativo que se escoja.
Teniendo en cuenta que únicamente vamos a desarrollar la aplicación para una plataforma, parece lógico que la ésta se desarrolle para el sistema operativo móvil más utilizado en la actualidad. Por ello, Android será el sistema operativo elegido para desarrollar la aplicación, ya que abarca la mayor cuota de mercado.
Así mismo, los lenguajes de programación específicos para la plataforma Android son C++, Java y XML o con Kotlin, de los cuales se posee un mayor nivel de conocimientos previos.
Además, se pretende crear una interfaz personalizable y versátil para lo cual Android es el sistema operativo adecuado ya que permite configurar el diseño de la pantalla de forma muy fácil y con mucha libertad.

Otro motivo de peso para desarrollar está aplicación en Android y no en iOS, su principal competidor, es el precio. Una cuenta de desarrollador de Apple (Apple Developer Program) cuesta 99 dólares por membresía al año. Para una empresa es de 299 dólares al año.
En cambio, el precio de creación de la cuenta en Google Play es de 25 dólares y se paga una sola vez. No hay cargos extras si quieres actualizar tu app Android en el futuro. Asimismo, puedes publicar varias apps de Android utilizando la misma cuenta de Desarrollador.
Además, iOS es un sistema operativo para el cual hace falta disponer de un terminal Mac (PCs de la marca Apple) y de un Iphone (terminal móvil de la marca Apple) para desarrollar la aplicación. Ambos terminales tienen un precio elevado.
Sin embargo, para desarrollar una aplicación para Android únicamente hace falta cualquier PC, y un móvil Android, que se puede encontrar a un precio mucho más asequible en el mercado.

## 4.3. Entornos integrados de desarrollo para Android

Como todos conocemos, el lenguaje que Google pensó que debería de usarse para programar aplicaciones para Android es Java. A continuación, realizaremos un repaso a los distintos entornos de desarrollo (IDE) disponibles y más populares por los cuales empezar a desarrollar aplicaciones para Android.

El concepto de IDE, que no es más que un programa informático compuesto por un conjunto de herramientas de programación. Es un entorno de programación empaquetado como un programa o aplicación, que nos provee de un marco de trabajo agradable para la mayoría de los lenguajes de programación. Constan entre sus características básicas con:

- Editor de código
- Compilador
- Depurador (debbuger)
- Constructor de interfaz gráfica

### 4.3.1. Eclipse

Eclipse es un entorno de desarrollo, de código abierto y gratuito, cuyo diseño sigue un patrón de actualización basado en plugins. Su objetivo es convertirse en una plataforma de integración de herramientas de desarrollo. Es un IDE que podríamos denominar genérico, ya que no fue concebido para ser utilizado con un solo lenguaje de programación, sino que es compatible con una gran variedad de lenguajes.

- Gestión de proyectos: El desarrollo sobre Eclipse se basa en proyectos.
- Depurador de código: Eclipse incluye un potente depurador de código, fácil e intuitivo, que nos proporciona de forma gráfica una opción de mejorar nuestros proyectos. Tiene

una perspectiva dedicada a la depuración donde se puede realizar y supervisar dicha tarea.
- Acceso a Base de Datos: Permite conectarse a distintos gestores de bases de datos y consultar tablas y datos. Aunque para ello hace falta un plugin de terceros.
- Perspectivas, editores y vistas: En Eclipse el concepto de trabajo se basa en las perspectivas, que son una configuración de ventanas y editores que nos permiten trabajar en un determinado entorno de trabajo de forma óptima.
- Colección de plugins: Hay una gran cantidad de plugins disponibles, algunos desarrollados por Eclipse y otros por terceros. Para poder desarrollar en Android, hace falta descargar plugins secundarios.
- Construcción de paquetes .apk mediante el uso de ANT: no permite importar proyectos creados en Android Studio, que a diferencia de Eclipse y Netbeans, utiliza Gradle.

### 4.3.2. Netbeans

NetBeans es un IDE que nos permite de forma rápida y fácil desarrollar aplicaciones Java de escritorio, móviles y aplicaciones web, utilizando tecnologías y lenguajes como HTML5 con HTML, Javascript y CSS. Proporciona un conjunto de herramientas para PHP, C y C++.

Entre sus puntos fuertes al igual que Eclipse es que ambos son de código abierto y gratuitos.
Se construye siguiendo la misma filosofía que Eclipse, ambos poseen una estructura modular para aumentar y configurar el entorno dependiendo de nuestras necesidades como desarrolladores.

- Asistentes y Gestor de Proyectos: El desarrollo sobre Netbeans se basa en proyectos. Tiene asistentes para configuración de distintos proyectos y selección de frameworks.
- Depurador de Código: el depurador permite, entre otras cosas, monitorizar en tiempo real los valores de las propiedades y variables.
- Acceso a Base de Datos y Plugins: Permite conectarse a distintos gestores de bases de datos y consultar tablas y datos.
- Construcción de paquetes .apk mediante el uso de ANT: no permite importar proyectos creados en Android Studio, que a diferencia de Eclipse y Netbeans, utiliza Gradle.
- Para poder desarrollar en Android, hace falta descargar plugins secundarios.

### 4.3.3. Intellij idea

Existen dos distribuciones disponibles de este IDE, una gratuita y de código abierto denominada IntelliJ IDEA Community Edition y otra de pago denominada IntelliJ IDEA Ultimate. Ambas son multiplataforma y están disponibles tanto para Windows como para Linux y Mac.
Las características principales de la versión gratuita son:

- Editor de Código: El editor resalta advertencias y errores inmediatamente. Además, permite finalización inteligente de código y autocompletado sofisticado y sensible.
- Herramientas integradas de Android: Posee un diseñador de interfaz de usuario que permite arrastrar y soltar elementos. También posee filtros de búsqueda personalizados.
- Aumento de productividad: Tiene soporte para Maven y Gradle, y un control de versiones compatible con Gib, GibHub y SVN, entre otros.
- Lenguajes soportados: Soporta varios lenguajes basados en JVM, como Java.

La versión de pago, entre otras cosas, añade un mayor soporte de lenguajes de programación, como PHP, SQL, MySQL.

### 4.3.4. Android Studio

Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de apps para Android, basado en IntelliJ IDEA. Además del potente editor de códigos y las herramientas para desarrolladores de IntelliJ, Android Studio ofrece incluso más funciones que aumentan tu productividad cuando desarrollas apps para Android, como las siguientes:

- Un sistema de compilación flexible basado en Gradle
- Un emulador rápido y cargado de funciones.
- Un entorno unificado donde puedes desarrollar para todos los dispositivos Android.
- Aplicación de cambios para insertar cambios de códigos y recursos a la aplicación en ejecución sin reiniciar la aplicación.
- Integración con GitHub y plantillas de código para ayudarte a compilar funciones de apps comunes y también importar código de muestra.
- Variedad de marcos de trabajo y herramientas de prueba.
- Herramientas de Lint para identificar problemas de rendimiento, usabilidad y compatibilidad de la versión, entre otros.
- Compatibilidad con C++ y NDK.
- Compatibilidad integrada con Google Cloud Platform, que facilita la integración con Google Cloud Messaging y App Engine.

### 4.3.5. Aide

AIDE se trata de un IDE para Android que nos permite programar y compilar nuestros proyectos directamente en nuestro dispositivo Android. Entre sus características principales nos ofrece la posibilidad de seguir programando nuestros proyectos directamente en nuestras tablets y poder emular directamente en el dispositivo la aplicación.

Dentro de las funcionalidades que podemos encontrar en AIDE, destacamos las siguientes:

- Comprobación de errores en tiempo real, autocompletado, formato y navegación inteligente además de contar con un análisis y resaltamiento de color del propio código.
- Posibilidad de programar aplicaciones directamente en una tablet.
- Compatibilidad total con proyectos de Eclipse.
- Completamente gratuito.

### 4.3.6. Elección del entorno de desarrollo

En cuanto al análisis de los principales entornos de desarrollo de aplicaciones para la plataforma Android, se ha llegado a la conclusión de que finalmente la elección para desarrollar esta aplicación será Android Studio.

La principal razón de peso por la cual se ha elegido Android Studio ha sido por la posibilidad de realizar un curso online “Introducción a la programación” ofertado por edX gracias a un convenio con la universidad. De este modo se ha podido aumentar los conocimientos previos sobre este entorno de desarrollo, así como el lenguaje de programación utilizado.
Esto ya supone una ventaja, ya que de los demás entornos de desarrollo no se tienen ningún conocimiento previo.

Se puede observar que entre los IDEs Eclipse y Netbeans tanto sus características como la instalación y el modo de uso, son bastante similares. En cuanto a IntelliJ IDEA se ha descartado como solución escogida ya que la versión gratuita tiene algunas limitaciones que podrían interferir en el desarrollo de la aplicación. En el entorno de desarrollo AIDE la aplicación se realiza en un dispositivo móvil Android, por tanto, esta opción es descartada debido a que el desarrollo completo de la aplicación de dicho modo sería muy complicado.

# 5. Android

En este apartado se procede a explicar con mayor detalle por qué Android ha sido el sistema operativo elegido, así como sus características y arquitectura.

## 5.1.¿Por qué Android?

Existen muchas plataformas para móviles (Apple iOS, Windows Phone, BlackBerry, Palm, Java Micro Edition, Linux Mobile (LiMo), Firefox OS, etc.); sin embargo, Android presenta una serie de características que lo hacen diferente. Es el primero que combina en una misma solución las siguientes cualidades:

- Plataforma abierta. Es una plataforma de desarrollo libre basada en Linux y de código abierto. Una de sus grandes ventajas es que se puede usar y customizar el sistema sin pagar royalties.
- Adaptable a diversos tipos de hardware. Android no ha sido diseñado exclusivamente para su uso en teléfonos y tabletas. Hoy en día podemos encontrar relojes, gafas, cámaras, TV, sistema para automóviles, electrodomésticos y una gran variedad de sistemas empotrados que se basan en este sistema operativo, lo cual tiene sus evidentes ventajas, pero también va a suponer un esfuerzo adicional para el programador. La aplicación ha de funcionar correctamente en dispositivos con una gran variedad de tipos de entrada, pantalla, memoria, etc. Esta característica contrasta con la estrategia de Apple: en iOS tenemos que desarrollar una aplicación para iPhone y otra diferente para iPad.
- Portabilidad asegurada. Las aplicaciones finales son desarrolladas en Java, lo que nos asegura que podrán ser ejecutadas en cualquier tipo de CPU, tanto presente como futuro. Esto se consigue gracias al concepto de máquina virtual.
- Arquitectura basada en componentes inspirados en Internet. Por ejemplo, el diseño de la interfaz de usuario se hace en XML, lo que permite que una misma aplicación se ejecute en un reloj de pantalla reducida o en un televisor.
- Filosofía de dispositivo siempre conectado a Internet. Muchas aplicaciones solo funcionan si disponemos de una conexión permanente a Internet. Por ejemplo, comunicaciones interpersonales o navegación con mapas.
- Gran cantidad de servicios incorporados. Por ejemplo, localización basada tanto en GPS como en redes, bases de datos con SQL, reconocimiento y síntesis de voz, navegador, multimedia, etc.
- Aceptable nivel de seguridad. Los programas se encuentran aislados unos de otros gracias al concepto de ejecución dentro de una caja, que hereda de Linux. Además, cada aplicación dispone de una serie de permisos que limitan su rango de actuación (servicios de localización, acceso a Internet, etc.). Desde la versión 6.0 el usuario puede conceder o retirar permisos a las aplicaciones en cualquier momento.
- Optimizado para baja potencia y poca memoria. En el diseño de Android se ha tenido en cuenta el hardware específico de los dispositivos móviles. Por ejemplo, Android utiliza la máquina virtual ART (o Dalvik en versiones antiguas). Se trata de una implementación de Google de la máquina virtual Java optimizada para dispositivos móviles.
- Alta calidad de gráficos y sonido. Gráficos vectoriales suavizados, animaciones, gráficos en 3D basados en OpenGL. Incorpora los codecs estándares más comunes de audio y vídeo, incluyendo H.264 (AVC), MP3, AAC, etc.

Android combina características muy interesantes. En definitiva, ¿por qué elegir Android para el desarrollo de la aplicación? Principalmente porque es el sistema operativo móvil más usado actualmente, además Android nos ofrece una forma sencilla y novedosa de implementar potentes aplicaciones para diferentes tipos de dispositivos.

Se muestra un gráfico en la figura 1 con el porcentaje de sistemas operativos más usados:

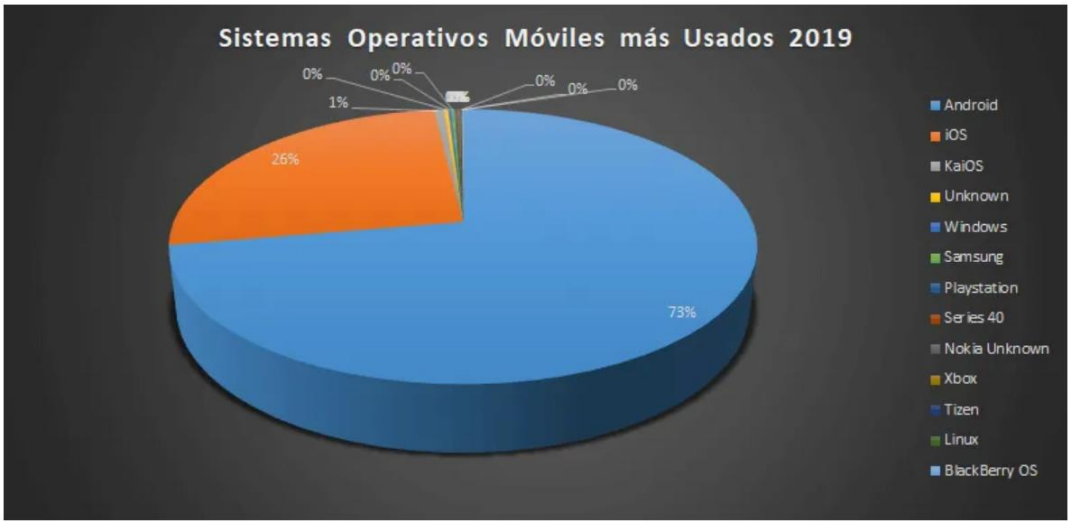


Figura 1: Sistemas Operativos móviles más usados en 2019. (Ramírez, 2019)

Durante el tercer trimestre de 2019 según el estudio de (Collado, n.d.) los usuarios de smartphones gastaron un total de 23.000 millones de dólares en apps y juegos, y una vez más, la diferencia entre las dos plataformas móviles vuelve a ser abismal.
Las estadísticas de consumo de aplicaciones y juegos de los poseedores de dispositivos Android y iOS a lo largo de los últimos tres meses nos ofrecen algunos datos curiosos que se ilustran en la figura 2, como el hecho de que en Google Play se descargaron un 175% más de apps y juegos que en la App Store de iOS.

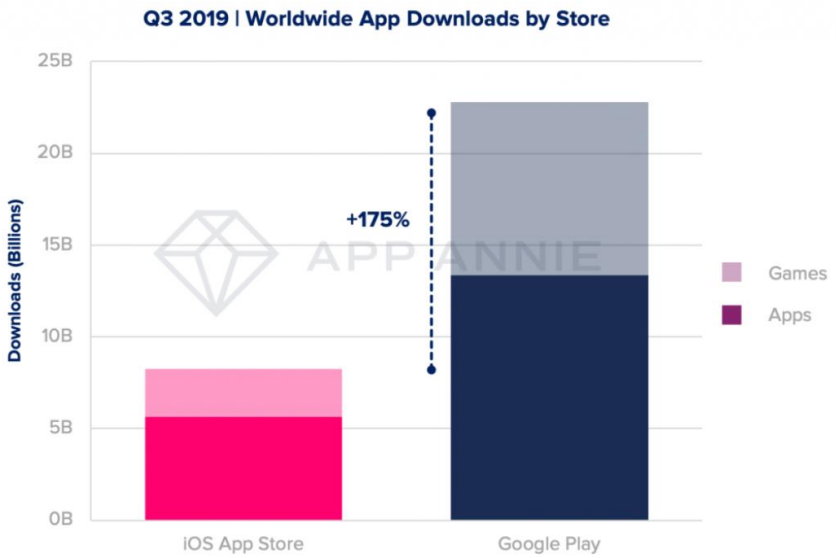


Figura 2: Aplicaciones descargadas por sistema operativo 2019.(Collado, n.d.)

Otro aspecto fundamental a la hora de comparar las plataformas móviles es su cuota de mercado. En la figura 3 se puede ver un estudio realizado por la empresa Gartner Group, donde se muestra la evolución del mercado de los sistemas operativos para móviles según el número de terminales vendidos. Podemos destacar la desaparición de la plataforma Symbian de Nokia, el declive continuo de BlackBerry, el estancamiento de la plataforma de Windows, que parece que no despega, y el afianzamiento de la cuota de mercado de Apple en torno al 15 %. En la gráfica se puede apreciar como Apple consigue anualmente un aumento significativo de ventas coincidiendo con el lanzamiento de un nuevo terminal. Finalmente, cabe señalar el espectacular

ascenso de la plataforma Android, que en cinco años ha alcanzado una cuota de mercado superior al 80 %.

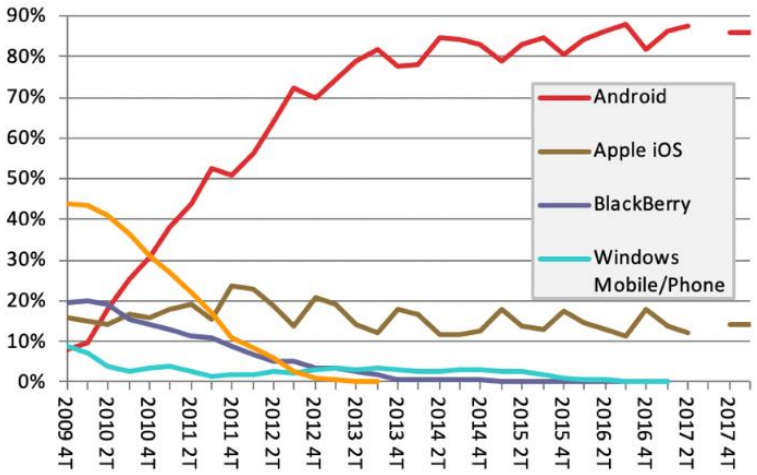


Figura 3: Porcentaje de teléfonos inteligentes vendidos en todo el mundo (*Máster en Desarrollo de Aplicaciones Android - Componentes de una aplicación*, 2017)

## 5.2. Versiones de Android

Antes de empezar a programar en Android hay que elegir la versión del sistema para la que deseamos realizar la aplicación. Es muy importante observar que hay clases y métodos que están disponibles a partir de una versión; si las vamos a usar, hemos de conocer la versión mínima necesaria.
A la hora de seleccionar la plataforma de desarrollo hay que consultar si necesitamos alguna característica especial que solo esté disponible a partir de una versión. Todos los usuarios con versiones inferiores a la seleccionada no podrán instalar la aplicación. Por lo tanto, es recomendable seleccionar la menor versión posible que nuestra aplicación pueda soportar.

Android no ha parado de evolucionar desde el lanzamiento de su primera versión. En febrero de 2009 Google lanza su primera actualización 1.1 que se diferenciaba de la primera en poder adjuntar archivos en los mensajes. Si echáramos la vista atrás y lo comparásemos con una versión actual, comprenderíamos la inmensa evolución del sistema operativo. Cuando se ha lanzado una nueva plataforma, siempre ha sido compatible con las versiones anteriores. Es decir, solo se añaden nuevas funcionalidades, y en el caso de modificar alguna funcionalidad, no se elimina, sino que se etiqueta como obsoleta, pero se puede continuar utilizando.
Todas las versiones de Android reciben del inglés el nombre de diferentes postres, siguiendo, además, orden alfabético. Las plataformas se identifican de tres formas alternativas: versión, nivel de API y nombre comercial. El nivel de API corresponde a números enteros, comenzando desde 1.

- **A: Apple Pie** (v1.0 nivel de API 1): tarta de manzana. (En septiembre de 2008)
- **B: Banana Bread** (v1.1 nivel de API 2): pan de plátano. En febrero de 2009.
- **C: Cupcake** (v1.5 nivel de API 3): panqué. En abril de 2009.
- **D: Donut** (v1.6 nivel de API 4): rosquilla. En septiembre de 2009.
- **E: Éclair** (v2.0/v2.1 nivel de API 5/7): pastel francés. En octubre de 2009 y en enero de 2010.
- **F: Froyo** (v2.2 nivel de API 8) (abreviatura de «*frozen yogurt*»): yogur helado. En mayo de 2010.
- **G: Gingerbread** (v2.3 nivel de API 9): pan de jengibre. En diciembre de 2010.

- **H: Honeycomb** (v3.0/v3.1/v3.2 nivel de API 11/12/13): panal de miel. En febrero, mayo y junio de 2011.
- **I: Ice Cream Sandwich** (v4.0 nivel de API 14): emparedado de helado. En octubre y diciembre de 2011.
- **J: Jelly Bean** (v4.1/v4.2/v4.3 nivel de API 16/17/18): gominola. En julio y noviembre de 2012 y en julio de 2013.
- **K: KitKat** (v4.4 nivel de API 19): tableta de chocolate con leche. En octubre de 2013.
- **L: Lollipop** (v5.0/v5.1 nivel de API 21/22): en noviembre de 2014 y en marzo de 2015
- **M: Mashmallow** (v6.0 nivel de API 23): en octubre de 2015
- **N: Nougat** (v7.0/v7.1 nivel de API 24/25): en julio y agosto de 2016
- **O: Oreo** (v8.0 nivel de API 26): en agosto de 2016
- **P: Pie** (v9.0 nivel de API 28): en agosto de 2018

Actualmente conviven muchas de las versiones mencionadas en el mercado, debido a que muchos terminales no se han actualizado a nuevas versiones. En la figura 4 se proporcionan datos sobre el porcentaje de terminales que usan las diferentes versiones.

| Version | Codename | API | Distribution |
|---|---|---|---|
| 2.3.3 - 2.3.7 | Gingerbread | 10 | 0.3% |
| 4.0.3 - 4.0.4 | Ice Cream Sandwich | 15 | 0.3% |
| 4.1.x | Jelly Bean | 16 | 1.2% |
| 4.2.x | | 17 | 1.5% |
| 4.3 | | 18 | 0.5% |
| 4.4 | KitKat | 19 | 6.9% |
| 5.0 | Lollipop | 21 | 3.0% |
| 5.1 | | 22 | 11.5% |
| 6.0 | Marshmallow | 23 | 16.9% |
| 7.0 | Nougat | 24 | 11.4% |
| 7.1 | | 25 | 7.8% |
| 8.0 | Oreo | 26 | 12.9% |
| 8.1 | | 27 | 15.4% |
| 9 | Pie | 28 | 10.4% |

Figura 4: Porcentajes de utilización de las versiones de Android (GoogleDevelopers, 2019)

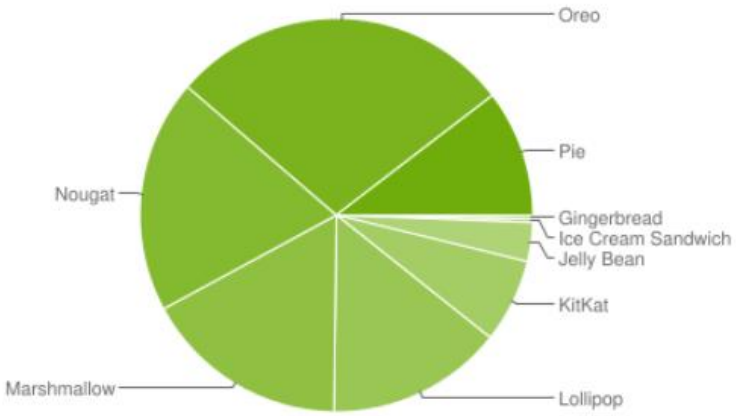


Figura 5: Versiones de Android más utilizadas (GoogleDevelopers, 2019)

## 5.3. Arquitectura Android

La arquitectura interna de Android está básicamente formada por 5 componentes o capas (Figura 6) que facilitan al desarrollador la creación de aplicaciones. Cada una de las capas utiliza elementos de la capa anterior para realizar sus funciones, es por ello por lo que a este tipo de arquitectura también se le conoce como *pila*. Por tanto, Android es una pila de software de código abierto basado en Linux creada para una variedad amplia de dispositivos y factores de forma. En el siguiente diagrama, se muestran los componentes principales de la plataforma Android.

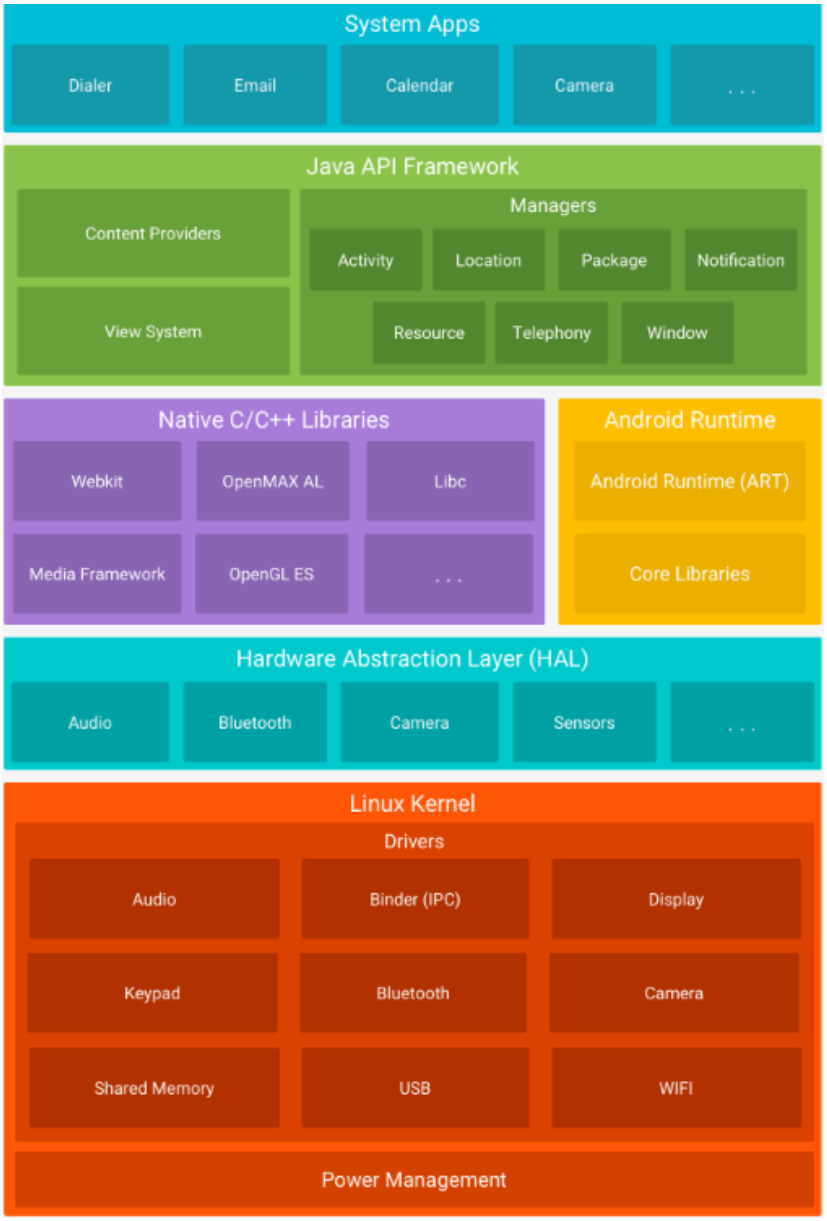


Figura 6: Pila de software de Android (Developers, 2020)

- Núcleo Linux

La base de la plataforma Android es el *kernel* de Linux. El núcleo de Android está formado por el sistema operativo Linux versión 2.6. Esta capa del modelo actúa como capa de abstracción entre el hardware y el resto de la pila. Por lo tanto, es la única que es dependiente del hardware. El desarrollador no podrá acceder directamente a esta capa, sino que debe utilizar las librerías disponibles en capas superiores. Para cada elemento de hardware del teléfono existe un controlador (o driver) dentro del *kernel* que permite su utilización desde el software.

El *kernel* también se encarga de gestionar los diferentes recursos del teléfono, proporciona servicios de seguridad, gestión de memoria, multiproceso, soporte para controladores de dispositivo y pila de red.

- Capa de abstracción del hardware (HAL)

La capa de abstracción de hardware (HAL) brinda interfaces estándares que exponen las capacidades de hardware del dispositivo al marco de trabajo de la API de Java de nivel más alto. La HAL consiste en varios módulos de biblioteca y cada uno de estos implementa una interfaz para un tipo específico de componente de hardware, como el módulo de la cámara o de Bluetooth. Cuando el marco de trabajo de una API realiza una llamada para acceder a hardware del dispositivo, el sistema Android carga el módulo de biblioteca para el componente de hardware en cuestión.

- Entorno de ejecución de Android

Está basado en el concepto de máquina virtual utilizado en Java. Dadas las limitaciones de los dispositivos donde ha de correr Android (poca memoria y procesador limitado), no fue posible utilizar una máquina virtual Java estándar. Google tomó la decisión de crear una nueva, la máquina virtual Dalvik, que respondiera mejor a estas limitaciones. A partir de Android 5.0 se reemplaza Dalvik por ART. Esta nueva máquina virtual consigue reducir el tiempo de ejecución del código Java hasta en un 33%.
Para los dispositivos con Android 5.0 (nivel de API 21) o versiones posteriores, cada app ejecuta sus propios procesos con sus propias instancias del tiempo de ejecución de Android (ART). El ART está escrito para ejecutar varias máquinas virtuales en dispositivos de memoria baja ejecutando archivos DEX, un formato de código de bytes diseñado especialmente para Android y optimizado para ocupar un espacio de memoria mínimo. Crea cadenas de herramientas, como Jack, y compila fuentes de Java en código de bytes DEX que se pueden ejecutar en la plataforma Android.

Estas son algunas de las funciones principales del ART:

- Compilación ahead-of-time (AOT) y just-in-time (JIT)
- Recolección optimizada de elementos no utilizados (GC)
- En Android 9 (nivel de API 28) y versiones posteriores, se convierten los archivos de formato ejecutable (DEX) de un paquete de aplicaciones a un código de máquina más compacto
- Esto mejora la compatibilidad con la depuración, el generador de perfiles de muestras dedicado, las excepciones de diagnóstico detalladas y los informes de fallos, y la capacidad de establecer puntos de control para supervisar campos específicos

Antes de Android 5.0 (nivel de API 21), Dalvik era el entorno de ejecución del sistema operativo. También se incluye en el runtime de Android el módulo Core Libraries, con la mayoría de las librerías disponibles en el lenguaje Java.

- Librerías

Estas librerías se han creado mediante C/C++ y compiladas en código nativo del procesador. Son junto con el núcleo los pilares que constituyen Android. Muchas librerías utilizan código abierto. Entre ellas se destacan:

- System C Library: adaptada para dispositivos embebidos en Linux.
- Media Framework: librería basada en OpenCORE de PacketVideo. Soporta codecs de reproducción y grabación de multitud de formatos de audio y vídeo e imágenes MPEG4, H.264, MP3, AAC, AMR, JPG y PNG.
- WebKit**:** Soporta el navegador web de Android y su vista webview. Misma librería que Google Chrome y Safari de Apple.
- SQLite: Ligero pero potente motor de bases de datos relacionales.
- SSL: Proporciona servicios de encriptación (*Secure Socket Layer*)

- Framework de Aplicaciones

Todo el conjunto de funciones del SO Android está disponible mediante API escritas en el lenguaje Java. Estas API son los cimientos que necesitas para crear apps de Android, el conjunto de herramientas de desarrollo. Todas las aplicaciones utilizan las mismas API y el mismo Framework. Ofrece una plataforma de desarrollo libre para aplicaciones, donde su principal riqueza reside en la reutilización de componentes. Componentes desarrollados por Google, o por usuarios. Los servicios más importantes son:

- Views: conjunto de vistas que sirven para compilar la IU de una app.
- Resources Manager: proporciona acceso a recursos sin código, como strings localizadas, gráficos, archivos de diseño.
- Location Manager: proporciona servicios de localización a aplicaciones.
- Activity Manager: manejador del ciclo de vida de las actividades.
- Notification Manager: permite mostrar notificaciones a las actividades.
- Content Provider: concede accesos a datos de otras aplicaciones

- Aplicaciones

En este nivel está formado por el conjunto de aplicaciones instaladas en una máquina Android tanto las nativas (programadas en C o C++) y las administradas (programadas en Java), las que añade el usuario y las que vienen por defecto con el sistema operativo. Están en el primer nivel por que utilizan todo lo posterior.

En Android se incluye un conjunto de apps centrales para correo electrónico, mensajería SMS, calendarios, navegación en Internet y contactos, entre otros elementos. Las apps del sistema funcionan como apps para los usuarios y brindan capacidades claves a las cuales los desarrolladores pueden acceder desde sus propias apps.

Normalmente las aplicaciones Android están escritas en Java o Kotlin. Para desarrollar este tipo de aplicaciones podemos utilizar el Android SDK. Existe otra opción consistente en desarrollar las aplicaciones utilizando C/C++. Para esta opción podemos utilizar el Android NDK (Native Development Kit).

# 6. Desarrollo del proyecto

En esta parte del proyecto se procede en primer lugar a dar una pequeña introducción sobre cuáles son los principales componentes de una aplicación, los ficheros y directorios de un proyecto Android y en concreto como se ha estructurado el presente proyecto.
Seguidamente se procede a la explicación del funcionamiento de la aplicación, mostrando cada una de sus ventanas, así como la interfaz que el usuario vería en cada momento.

En este apartado comprobamos que el programa cumple los requisitos del proyecto.
Posteriormente se muestra el desarrollo del código más importante de la aplicación para cada una de las partes del proyecto, explicando a su vez el funcionamiento de éste.

## 6.1. Introducción a Android

En este apartado se muestran los componentes básicos de una aplicación Android y la estructura de directorios que ha de tener un proyecto desarrollado en este sistema operativo.

### 6.1.1. Componentes de una aplicación Android

Los principales componentes de una aplicación Android son los siguientes:

- Actividad (*Activity*): cada una de las pantallas de la aplicación es una actividad. Cada aplicación suele estar formada por varias actividades, que en conjunto crean la interfaz de usuario. Es el medio de comunicación entre la aplicación y el usuario. Los elementos que se muestran en ella deben ser definidos en el fichero xml que llevan asociado, en el que se definen los elementos tales como ubicación de los elementos en la pantalla.

- Fragmento (*Fragment*): porción de interfaz de usuario que puede añadirse o quitarse de la interfaz, independientemente del resto de elementos de la actividad. En el caso del presente proyecto no se han utilizado fragmentos.

- Intención (*Intent*): representa la intención de realizar una acción. Se utiliza para lanzar una actividad o servicio. También permite comunicarse con un servicio o intercambiar información entre componentes lanzados.

- Layout: estos elementos, descendientes de la clase View, agrupan un conjunto de vistas de una determinada forma. Hay diferentes tipos de *Layout* en función de cómo se quieran organizar las vistas. Algunos de estos son:
    - Lineal: organiza sus hijos en una única fila, que pueden ser vertical u horizontal.
    - Relativo: permite especificar la posición d ellos objetos hijo en relación al padre o a otros objetos hijo.
    - Web: permit3e mostrar fácilmente páginas web.
    - Listview: define una columna con desplazamiento, es decir, una lista.
    - Gridview: define una cuadrícula de filas y columnas con desplazamiento.

- Proveedor de contenidos (*Content Provider*): permite a las aplicaciones compartir datos con otras aplicaciones sin comprometer la seguridad de su sistema de ficheros.

- Servicios (*Service)*: proceso que se ejecuta como actividad secundaria, sin necesidad de interacción con el usuario. Son componentes sin interfaz gráfica que se ejecutan en segundo plano, por ejemplo, actualizar datos, lanzar notificaciones, mostrar elementos visuales…

- Vista (*View)*: estos elementos, descendientes de la clase View, componen la interfaz de usuario de una aplicación. Habitualmente se definen en un fichero XML, aunque también pueden ser definidos en un .java.

### 6.1.2. Ficheros y directorios de un proyecto Android

Android tiene una estricta estructura de directorios, que a su vez permite tener la documentación de cualquier proyecto ordenada siempre del mismo modo.

Un proyecto en Android Studio puede contener varios módulos. Cada módulo corresponde a una aplicación o ejecutable diferente, útil cuando se quiera crear varias versiones de la aplicación. Cada módulo en Android está formado por un descriptor de la aplicación (manifests), el código fuente en Java (java), una serie de ficheros con recursos (res) y ficheros para construir el módulo (Gradle Scripts).

- *AndroidManifest.xml*: describe los componentes de la aplicación, así como sus características principales. Se define su nombre, paquete, icóno, estilos, etc. Se indican las actividades, las intenciones, los servicios y los proveedores de contenido de la aplicación. También se declaran los permisos que requerirá la aplicación. Se indica el paquete Java, la versión de la aplicación, etc.
- *Java:* Carpeta que contiene el código fuente de la aplicación. Los ficheros java se almacenan en carpetas según el nombre de su paquete.
    - *MainActivity:* Clase con el código de la actividad inicial.
    - *ExampleInstrumentTest:* Clase pensada para insertar código de testeo de la aplicación.
    - *ExampleUnitTest:* Clase para insertar test unitarios sobre otras aplicaciones.
- *Res/:* directorio que contiene los recursos de la aplicación. Está formado por varios directorios, cada uno de los cuales tiene una función.
    - *Drawable/:* directorio para imágenes (jpg y png) y descripciones de imágenes (XML). Los nombres de los archivos almacenados en esta carpeta solo pueden contener números y letras de la “a” a la “z” en minúscula, excluyendo la “ñ”.
    - *Layout/:* directorio para los ficheros que describen la interfaz de usuario de la aplicación.
    - *Menu/:* directorio para los ficheros XML con los menús de cada actividad.
    - *Mipmap/*: en esta carpeta se guarda el icono de la aplicación, funciona exactamente igual que la carpeta *drawable* pero se ha añadido en seis versiones diferentes para diferenciar los recursos en base a la densidad que les corresponde.
    - *Anim/* y *animator/*: directorio para ficheros XML relacionados con animaciones.
    - *Raw/*: ficheros adicionales que no se encuentran en formato XML.
    - *Values/*: directorio para los ficheros que contienen colecciones de recursos. Algunos de estos ficheros son:
        - Colors.xml: fichero que define los distintos colores usados en la aplicación.
        - Strings.xml: fichero que define las distintas cadenas (textos) usados en la aplicación. Se puede definir uno distinto para cada idioma.
        - Styles.xml: fichero que define los distintos estilos usados en la aplicación.
    - XML/: directorio para otros ficheros XML requeridos por la aplicación.
- Gradle Scripts/: en esta carpta se almacenan una serie de ficheros Gradle que permiten compilar y construir la aplicación. Observa como algunos hacen referencia al módulo app y el resto son para configurar todo el proyecto. El fichero más importante es build.gradle (Module:app) que es donde se configuran las opciones de compilación del módulo.

### 6.1.3. Paquetes de la aplicación

Los proyectos Android están divididos en diferentes carpetas, pero internamente se denominan

paquetes, y es donde se encuentran todos los recursos del proyecto. En estos paquetes hay unos con los ficheros de código java, otros con los archivos xml de visualización y configuración, y también otros recursos como las imágenes e iconos de la aplicación como se ha explicado en el apartado anterior.
A continuación, se muestra la estructura de paquetes resultante de la aplicación "CunicolApp":

En el caso de este proyecto se han dividido los ficheros de código .java en subcarpetas:

- *Io*: hace referencia a Input / Output, lo relacionado a consultar y escribir datos en el servidor.
- *Model*: modelado de datos, en este paquete se encuentran las clases del proyecto que representan las tablas de la base de datos.
- *Ui*: hace a referencia a User Interface donde se ubican las actividades y los fragmentos. Dentro de la carpeta Ui hay una subcarpeta llamada *adapter*, donde se encuentran los adaptadores del proyecto.

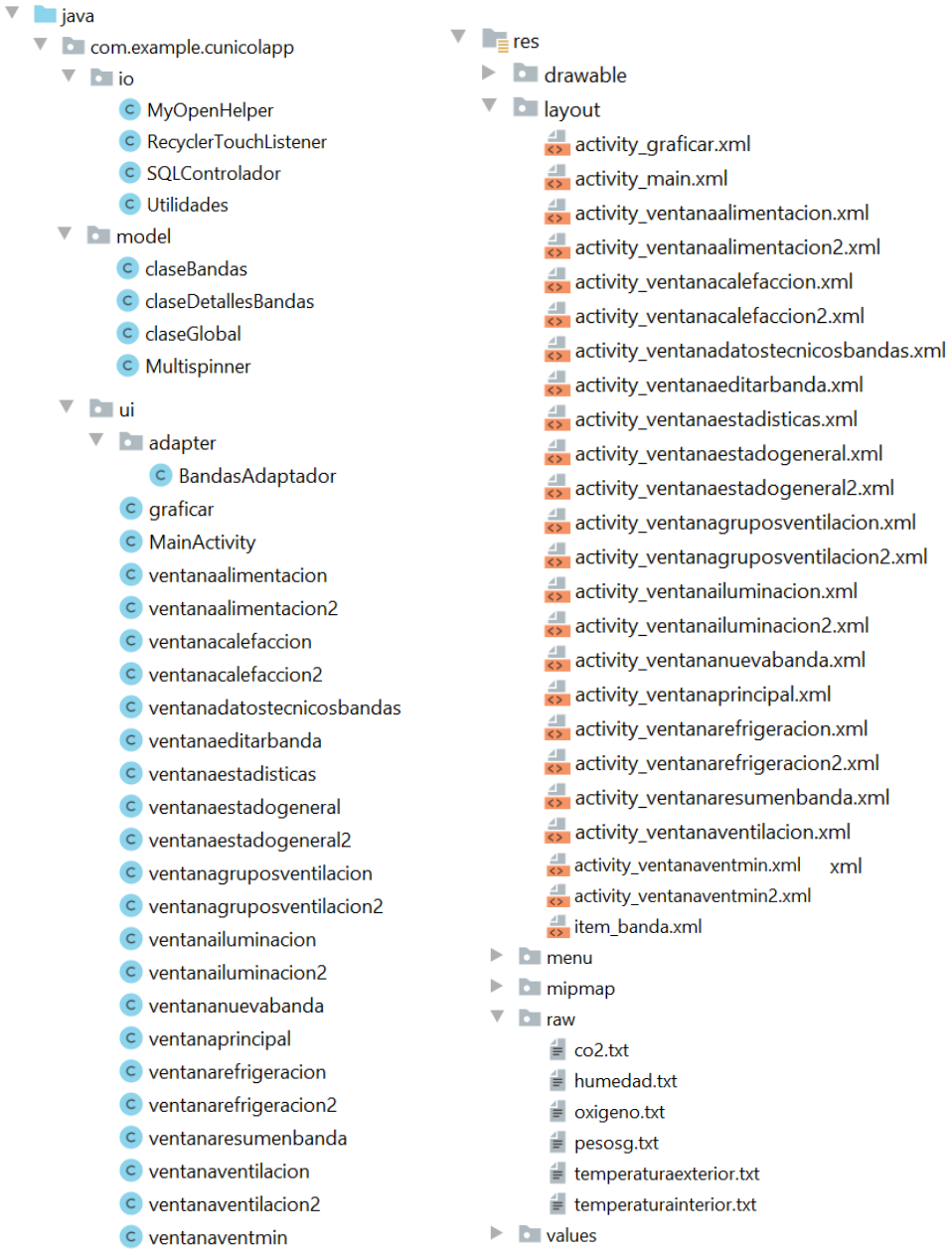


Figura 7: Captura de los paquetes y carpetas de la aplicación CunicolApp

## 6.2. Análisis de la aplicación

A continuación, se explica con detalle cómo funciona cada una de las diferentes ventanas desarrolladas en la aplicación. Cabe destacar que como la explotación está compuesta por dos naves independientes, tendremos una ventana de estado general, iluminación, ventilación, calefacción, refrigeración y alimentación para cada una de las naves, desde cada una de las cuales será posible acceder a la otra nave con pulsar un botón situado en la parte inferior derecha de la pantalla. Sin embargo, las ventanas de datos técnicos y estadísticas son comunes a la explotación, ya que contienen datos de ambas naves.

Además, es importante recordar que el presente proyecto es un prototipo, por lo tanto, debe cumplir las especificaciones de la aplicación real, pero introduciendo los datos que realmente serán arrojados por las sondas a modo de simulación.

Otro factor importante que se debe comentar es que la aplicación está diseñada para que su visualización en la pantalla sea horizontal. Esto se debe a que de este modo será más fácil para el usuario la navegación en la aplicación, así como la visualización de las diferentes ventanas y gráficas.

Este apartado del proyecto puede utilizarse a su vez como manual de usuario, ya que a continuación se explican todas las ventanas, los datos que se deben introducir por parte del usuario y qué es lo que debe ocurrir en cada ocasión.

En primer lugar, se procede a dar una explicación general de cómo funciona la aplicación en cuanto a la duración del ciclo y las tareas a realizar durante el mismo:

| | |
|---|---|
| ENTRAN CONEJAS | DIA 0 |
| PARTOS | DIA 7 |
| FLUSHING | DIA 11 |
| INSEMINACIÓN | DIA 18 |
| DESTETE | DIA 42 |
| VENTA | DIA 77 |
| VACIA | hasta el DIA 84 |

Figura 8: Cronología del ciclo de la explotación

En la figura 8 podemos observar cuales son las tareas a realizar en una explotación cunícola. Para todas las ventanas de la aplicación el ciclo comienza cuando se destetan los gazapos de las madres, es decir, el día cero del ciclo las conejas que en ese momento están preñadas, pasan a estar en una de las naves ellas solas.

Una semana después del destete, que correspondería con el día 7 del ciclo, las conejas empiezan a parir conejos de una nueva banda.
El día 11 postparto se inseminan las conejas y como ya se ha mencionado anteriormente, una semana antes deberá iniciarse el “flushing”.
El día 11 postparto corresponde con el día 18 del ciclo, por tanto, una semana antes del día 18 se debe iniciar el “flushing”. Será día 11 del ciclo cuando comenzará el intervalo de mayor tiempo de luz.

Generalmente se destetan los conejos en el día 42 del ciclo (momento en el que se quedarán en la nave solo los gazapos), pero este parámetro depende del ganadero y puede variar. Desde que se destetan los conejos de las madres, éstos estarán en la fase de engorde hasta que tengan 70 días de vida y llegue el momento de la venta. Del día 77 al 84 la nave estará vacía, y será cuando se realicen las labores de limpieza y desinfección para preparar la nave para la siguiente banda.

### 6.2.1. Login Activity

La ventana que se muestra a continuación es la primera interfaz que le aparecerá al usuario cuando inicie la aplicación. En ella se encuentra el nombre de la aplicación, el icono de ésta, así como dos apartados con sus respectivos iconos, en los que se deben introducir el usuario y la contraseña para acceder a la aplicación.

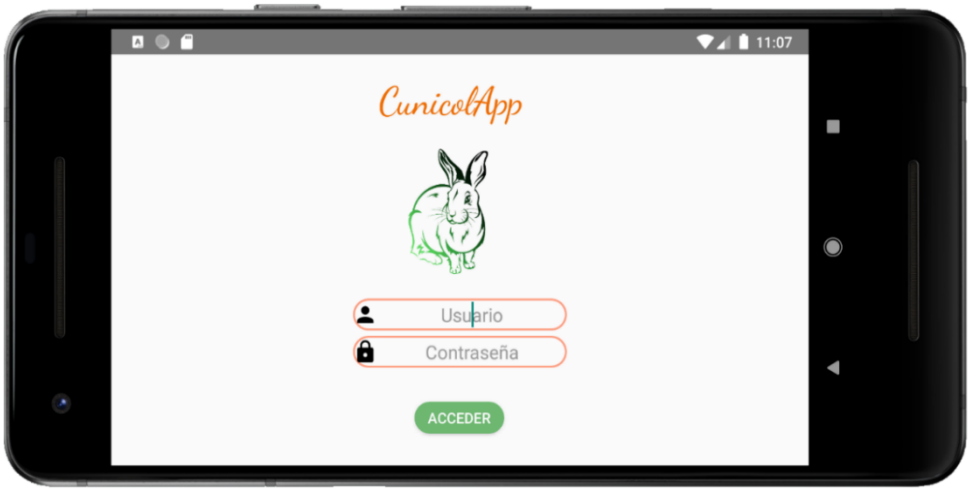


Figura 9: Ventana de Inicio

Si las credenciales que se introducen son incorrectas, aparecerá un mensaje que avisará al usuario de que el nombre de usuario o la contraseña no son los correctos. Este mensaje se muestra en la figura 10.

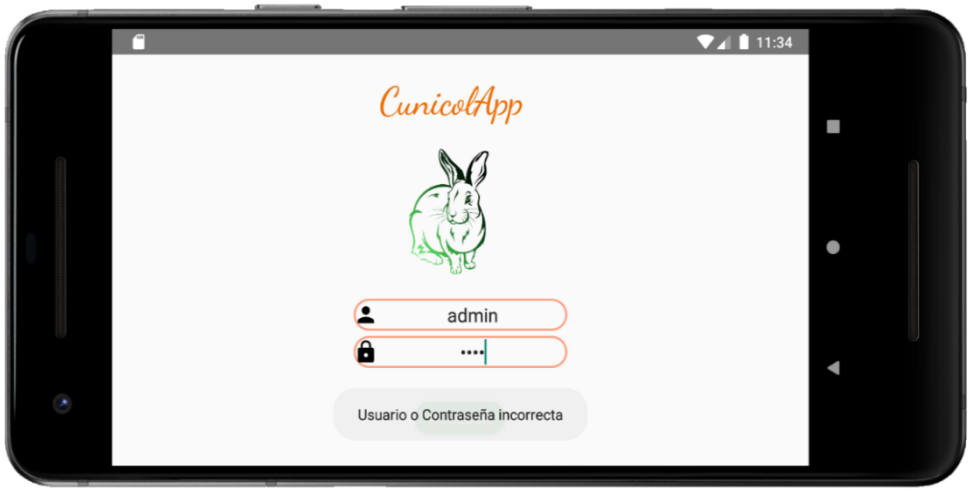


Figura 10: Ventana de inicio con credenciales incorrectas

### 6.2.2. Menú Principal

Tras introducir el usuario y la contraseña correctas, el usuario accederá al menú principal de la aplicación, que se muestra en la figura 11. Esta ventana se fracciona en ocho partes, se compone de un *GridLayout* en el cual se encuentran ocho *CardViews*, uno para cada ventana de la aplicación.


Figura 11: Ventana principal

Como ya se ha comentado anteriormente el proyecto trata dos naves independientes. Si el *Cardview* tiene tres puntitos en su parte superior quiere decir que hay una ventana para cada nave. Sin embargo, las dos últimas ventanas son comunes a ambas explotaciones. Para poder acceder a cada una de las naves se debe clicar en los tres puntitos o en cualquier parte del *Cardview* y aparecerá el siguiente menú sobre de la ventana seleccionada:

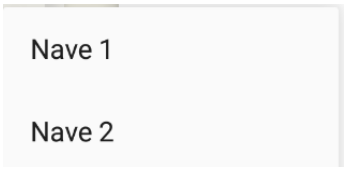


Figura 12: Menú de elección de nave (ventana principal)

### 6.2.3. Estado General

En el menú principal de la aplicación, la primera ventana que nos aparece en la parte superior izquierda de la pantalla es la ventana de estado general.
En esta pantalla el usuario puede comprobar el estado de los diferentes actuadores a controlar en la explotación de forma rápida y sencilla:


Figura 13: Ventana estado general

- La iluminación, que estará activa si la bombilla se encuentra encendida.
- El “cooling”, que estará en funcionamiento cuando aparezcan gotitas de agua en su parte superior e inferior.
- El cañón, que estará calentando cuando la imagen se muestre a color.
- Los cuatro ventiladores regulados y el ventilador de apoyo, que estarán activos cuando se muestren a color cada uno de ellos.
- Los silos, estarán accionados cuando el dibujo de los tres silos se muestre a color como en el caso de la figura 13.

Además, el usuario puede acceder desde esta pantalla a las diferentes ventanas de la aplicación con solo clicar los elementos descritos anteriormente.
En la parte superior derecha de la pantalla se muestra el día del ciclo en el que se encuentra la explotación.
También se muestran en esta ventana los datos de temperatura interior y exterior arrojados por las sondas de temperatura.

### 6.2.4. Iluminación

El programa de iluminación funciona como se explica a continuación y su interfaz se muestra en la siguiente imagen (figura 14).


Figura 14: Ventana de Iluminación

El usuario debe introducir la duración normal y la duración del “flushing”, así como la hora de inicio a la cual se encenderá el sistema. El programa calcula automáticamente la hora a la cual el sistema debe ser apagado.

Para asegurarnos de que el usuario introduce las horas y las duraciones en el formato correcto y además facilitar la introducción de los datos, se ha implementado un *TimePicker* como el de la figura 15 que aparecerá cada vez que el usuario quiera modificar alguno de dichos parámetros.

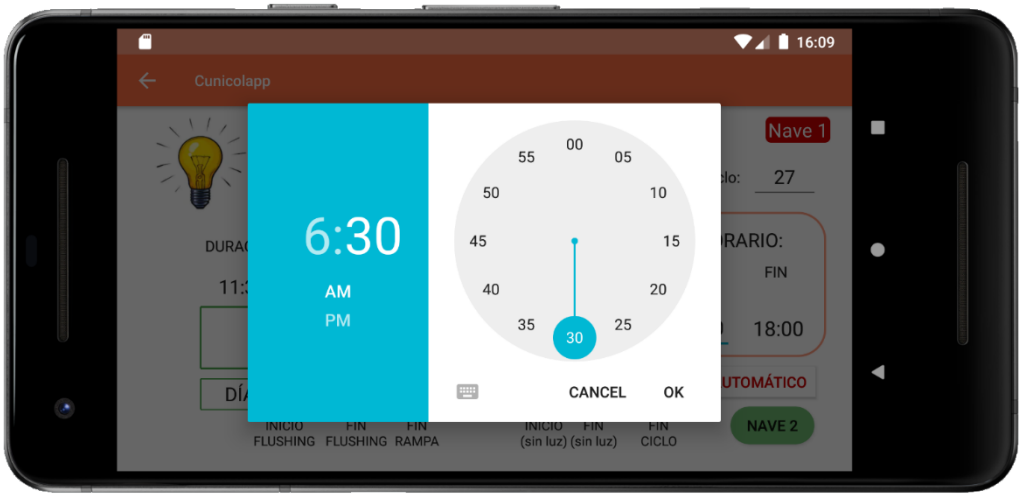


Figura 15: TimePicker (Ventana de iluminación)

Se muestra una gráfica para visualizar lo que ocurre en cada momento en la nave y se colorea de verde el tramo en el que se encuentra actualmente la explotación según el día del ciclo.

Como se ha explicado en la introducción de este apartado, el día 11 del ciclo comenzará el intervalo de mayor tiempo de luz, ya que se debe realizar el "flushing" una semana antes de inseminar.
Todos estos parámetros son editables para que el usuario adecue el programa de iluminación a sus necesidades. Además, se puede observar en qué día del ciclo se encuentran las naves, así como la hora actual y si las luces están encendidas o apagadas.

La ventana iluminación tiene un modo manual y otro automático. En el modo automático el programa será el encargado de comprobar si la iluminación debe estar encendida o apagada, en cambio en el modo manual, el usuario podrá encender y apagar la iluminación de la explotación en cualquier momento haciendo clic en la imagen de la bombilla, independientemente de la hora que sea.

### 6.2.5. Ventilación

En cuanto a la ventilación, además de introducir los parámetros para el cálculo de peso total de conejos, es decir, la cantidad de madres y gazapos que se encuentran en la nave, el usuario debe establecer manualmente unos parámetros:

- El valor mínimo y máximo de metros cúbicos / kilos de peso vivo necesarios para ventilar la nave. Estos valores varían y son diferentes para verano e invierno, pero normalmente están estipulados. Estos parámetros serán más bajos en invierno, ya que como el aire es frío se pretende ventilar lo mínimo para eliminar las sustancias de $CO_2$ y amoniacos presentes en el aire y así conseguir un buen ambiente. En verano no importa si ventilamos algo más por eso los valores son más altos.
- También se deben introducir las temperaturas mínima y máxima asociadas al mínimo y máximo de $m^3$de aire a ventilar.

El programa trabajará en la zona mínima, en la zona de regulación o en la zona máxima dependiendo de la temperatura de referencia que se obtenga, de las temperaturas mínima y máxima introducidas por el usuario y de los kilos de conejos que haya en cada momento en la explotación.

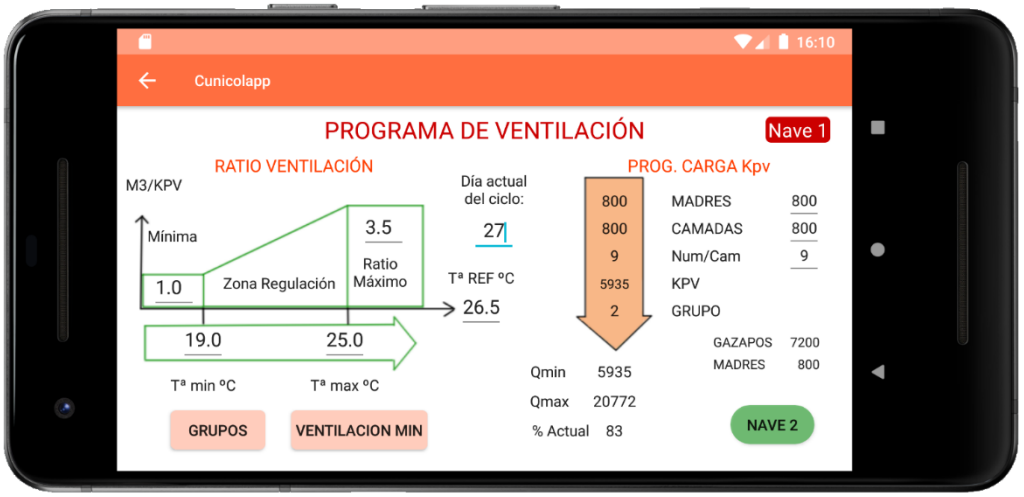


Figura 16: Ventana de Ventilación

El programa de ventilación funciona de la siguiente manera:

Una vez introducido todos los parámetros por parte del usuario, el programa calcula la ventilación necesaria en cada momento en función de los kilos de peso vivo existentes en la explotación.
En la flecha naranja se puede observar la cantidad de madres, camadas, gazapos por camada, kilos de peso vivo, y el grupo de ventilación que está funcionando.
Debajo de la flecha se observan los valores mínimo y máximo de metros cúbicos a ventilar, así como el porcentaje de ventilación actual al que está funcionando el grupo correspondiente.

Para realizar el cálculo del peso total partimos de una curva que ha sido facilitada por la empresa productora de la genética con la que se trabaja, de la cual obtenemos el peso estimado de un gazapo cada semana de crecimiento:

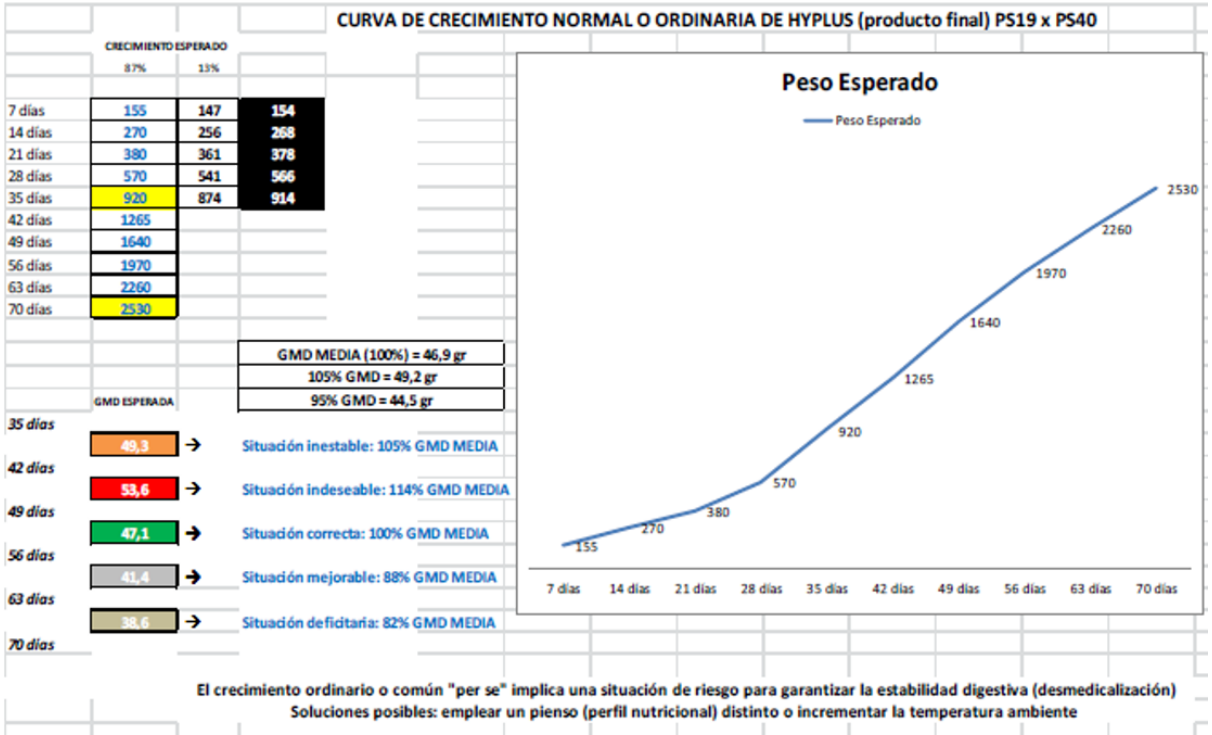


Figura 17: Curva de crecimiento normal u ordinaria de Hyplus

A continuación, se calculará el peso estimado de un gazapo cada día de crecimiento:

Según la información recogida, en una semana el gazapo tiene que pesar alrededor de 155 g. Si partimos de que el día en el que nacen los gazapos pesan aproximadamente 50 g, debemos dividir 105 gramos entre 6 días, lo que nos da como resultado que durante la primera semana los gazapos crecerán al día unos 17,5 gramos. Así sucesivamente con todos los datos que tenemos, hasta que los gazapos lleguen a los 70 días de vida, día en el cual deben pesar alrededor de 2,5 kilos.

| | |
|---|---|
| 1 | 50 |
| 2 | 67,5 |
| 3 | 85 |
| 4 | 102,5 |
| 5 | 120 |
| 6 | 137,5 |
| 7 | 155 |
| 8 | 171,43 |
| 9 | 187,86 |
| 10 | 204,29 |
| 11 | 220,72 |
| 12 | 237,15 |
| 13 | 253,58 |
| 14 | 270,01 |
| 15 | 285,72 |
| 16 | 301,43 |
| 17 | 317,14 |
| 18 | 332,85 |
| 19 | 348,56 |
| 20 | 364,27 |
| 21 | 379,98 |
| 22 | 407,12 |
| 23 | 434,26 |
| 24 | 461,4 |
| 25 | 488,54 |

| | |
|---|---|
| 26 | 515,68 |
| 27 | 542,82 |
| 28 | 569,96 |
| 29 | 619,96 |
| 30 | 669,96 |
| 31 | 719,96 |
| 32 | 769,96 |
| 33 | 819,96 |
| 34 | 869,96 |
| 35 | 919,96 |
| 36 | 969,25 |
| 37 | 1018,54 |
| 38 | 1067,83 |
| 39 | 1117,12 |
| 40 | 1166,41 |
| 41 | 1215,7 |
| 42 | 1264,99 |
| 43 | 1318,56 |
| 44 | 1372,13 |
| 45 | 1425,7 |
| 46 | 1479,27 |
| 47 | 1532,84 |
| 48 | 1586,41 |

| | |
|---|---|
| 49 | 1639,98 |
| 50 | 1687,12 |
| 51 | 1734,26 |
| 52 | 1781,4 |
| 53 | 1828,54 |
| 54 | 1875,68 |
| 55 | 1922,82 |
| 56 | 1969,96 |
| 57 | 2011,39 |
| 58 | 2052,82 |
| 59 | 2094,25 |
| 60 | 2135,68 |
| 61 | 2177,11 |
| 62 | 2218,54 |
| 63 | 2259,97 |
| 64 | 2298,54 |
| 65 | 2337,11 |
| 66 | 2375,68 |
| 67 | 2414,25 |
| 68 | 2452,82 |
| 69 | 2491,39 |
| 70 | 2529,96 |

Figura 18: Peso estimado de los gazapos cada día del ciclo

En la figura 18 podemos identificar cuanto debe pesar un gazapo según en el día del ciclo en el que se encuentre. La vida de un gazapo en la granja es de aproximadamente 70 días. Sin embargo, como ya se ha comentado anteriormente, el ciclo comienza cuando los conejos son destetados de las madres, es decir, las conejas (las cuales están preñadas) pasan a estar de nuevo en una nave ellas solas. A los 7 días de haber realizado el destete, las conejas parirán conejos de una nueva banda, ese es el día 7 del ciclo que corresponde con el día 1 de la figura 18, donde el gazapo acaba de nacer y pesará alrededor de 50 g.

El número de kilos vivos de carne se calcula de la siguiente manera:

El usuario introduce el número de madres, cuyo peso se ha estimado en cuatro kilos, el número de camadas y el número de conejos por camada.
La cantidad de gazapos se obtiene multiplicando el número de camadas por el número de conejos que hay en cada camada.
El resultado del número de madres y gazapos se muestra en la esquina inferior derecha de la ventana (figura 16).

Una vez que tenemos la información de los kilos totales existentes podemos calcular los $m^3/h$ de ventilación necesaria.
La ventilación mínima y máxima dependen del índice de ratio mínimo y máximo, los cuales son introducidos por el usuario, y se calculan de la siguiente manera:

$$q_{min} = ratio_{min} \cdot kg_{peso\ vivo}$$
$$q_{max} = ratio_{max} \cdot kg_{peso\ vivo}$$

El porcentaje de ventilación actual se calcula en función de los siguientes parámetros:

- $q_{min}$
- $q_{max}$
- $T^{a}_{min}$
- $T^{a}_{max}$
- $T^{a}_{interior}$

Cálculo de la pendiente de la recta:

$$m = \frac{q_{\ max} - q_{min}}{T^{a}_{max} - T^{a}_{min}}$$

Este valor corresponde a el incremento de ventilación(q) por grado (ºC).
Ecuación de la recta:

$$q_{\ actual} = m \cdot (T^{a}_{interior} - T^{a}_{min}) + q_{min}$$

Según los distintos parámetros que intervienen en este proceso los ventiladores encargados de realizar la tarea serán unos u otros en función de la necesidad de aire a ventilar, por tanto, según el grupo de ventiladores que actúa los porcentajes de ventilación a calcular son distintos en función de la capacidad de extracción de cada grupo de ventilación (qmax).

$$\%\ q_{min} = \frac{q_{min}}{q_{\max Grupo}} \cdot 100$$

$$\%\ q_{max} = \frac{q_{max}}{q_{\max Grupo}} \cdot 100$$

$$\%\ q_{actual} = \frac{q_{actual}}{q_{\max Grupo}} \cdot 100$$

El programa de ventilación activará el “cooling” cuando la temperatura interior de la nave supere en 1,5 ºC la temperatura máxima introducida por el usuario. En ese momento aparecerá un mensaje de aviso al usuario que le informará de la situación y le permitirá a su vez dirigirse hacia la pantalla de refrigeración.


Figura 19: Ventana de Ventilación. Alarma “cooling”.

Desde la ventilación se puede acceder a la ventana de ventilación mínima en la que se muestra lo siguiente:


Figura 20: Ventana de ventilación (mínima)

Los valores de ventilación mínima y máxima, la ventilación actual por $m^3/h$, así como dichos valores en porcentaje. También se puede observar en esta pantalla la temperatura de referencia y los valores de temperatura mínima y máxima introducidos por el usuario.

Desde la ventilación también podemos acceder a la pantalla de grupos en la que se distingue entre grupo de apoyo y grupos.


Figura 21: Ventana de ventilación (grupos)

La ventilación funciona por grupos de extracción. En la explotación se encuentran cuatro extractores de 12500 $m^3/h$ cada uno. Con estos extractores se forman cuatro grupos de ventilación que funcionan de la siguiente manera:
Los grupos actúan de menor a mayor porcentaje de ventilación según la necesidad de aire a ventilar. En primer lugar, entra en funcionamiento el grupo uno que corresponde al ventilador 1 (12500 $m^3/h$), cuando la ventilación necesaria supera la proporcionada por el grupo 1, entra en funcionamiento el grupo 2 el cual corresponde a los ventiladores 1 y 2 (25000 $m^3/h$) en el porcentaje correspondiente para completar los metros cúbicos necesarios. Lo mismo sucede con el grupo 3, ventiladores 1,2 y 3 (37500 $m^3/h$) y por último entraría en funcionamiento grupo 4, ventiladores 1,2,3 y 4 (50000 $m^3/h$) que corresponde con la capacidad máxima regulada de la nave.

A la derecha de la pantalla, en la sección de grupos, se puede ajustar los metros cúbicos que tienen los ventiladores, estos valores corresponden con la capacidad máxima de extracción de cada grupo ($q_{max}$), y son variables para que en el caso de que el usuario cambie de ventiladores pueda introducir los nuevos parámetros. Además de los cuatro grupos, también se utiliza un grupo de apoyo, que pasará a estar en funcionamiento cuando los ventiladores y el "cooling" no sean suficiente para enfriar las naves.

El grupo de apoyo funciona de la siguiente manera:
Cuando la ventilación regulada llega a su máximo, el "cooling" está encendido y la temperatura no desciende lo suficiente (esto ocurre normalmente en verano), entra en funcionamiento un grupo de apoyo, formado por un solo ventilador de 25000 $m^3/h$ situado en la parte superior de la nave. Este dejará de funcionar cuando la temperatura de la nave alcance la temperatura de corte del grupo de apoyo, siguiendo entonces la ventilación de forma regulada.

El usuario debe establecer el porcentaje mínimo y máximo al cual trabajará el ventilador, la temperatura de arranque, la temperatura máxima, la temperatura de alarma y los grados de histéresis de corte que servirán para calcular la temperatura de corte. Además, en la parte superior de la ventana se muestra el porcentaje de ventilación al cual estará trabajando el grupo de apoyo cuando este activo.

Por lo tanto, en la ventana principal de ventilación, cuando la temperatura interior de la nave alcance la temperatura mínima establecida en el grupo de apoyo, aparecerá un mensaje de alarma que avisará al usuario de que el grupo de apoyo ha sido encendido, además también se permite al usuario acceder a dicha ventana como se muestra en la figura 22.


Figura 22: Ventana de ventilación. Alarma grupo de apoyo.

### 6.2.6. Calefacción

El método de funcionamiento del sistema de calefacción es como un termostato, es decir, se establece una temperatura de referencia y una histéresis. El usuario introducirá un tercer parámetro que hace referencia al retardo en segundos que tarda el cañón en encenderse y en apagarse.

Por lo que el usuario debe establecer tres parameros:
La temperatura de referencia o temperatura de termostato y los grados de histéresis, determinan a que temperatura se encenderá el cañón. Lo que debe ocurrir es que cuando la temperatura de la nave baje a la temperatura de referencia, el cañón de propano se pondrá en marcha. Este dejará de actuar cuando la temperatura de la nave supere en la histéresis programada la temperatura de referencia.

Además, los cañones no se encienden y apagan al instante, sino que el usuario puede introducir un retardo en segundos para controlar esta situación. En el programa de calefacción se mostrará una imagen de un cañón de propano a color cuando esta deba estar encendida, y un mensaje como el de la figura 23 cuando esté apagada.

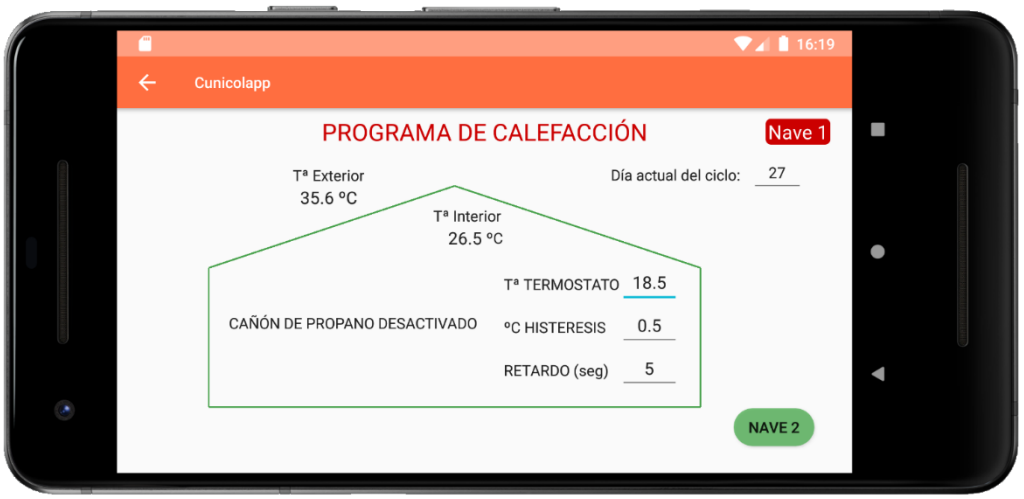


Figura 23: Ventana de Calefacción

## 6.2.7. Refrigeración

La refrigeración consiste en determinar una temperatura de referencia y gracias a una sonda de temperatura interior podremos saber cuándo la temperatura de la nave excede la de referencia, en ese momento se deberá accionar la bomba que moja el panel humidificador para que enfríe el aire que pasa a través de él.

El usuario deberá introducir también unos parámetros para controlar cuando es necesario que arranque, cuanto tiempo debe estar en marcha y cuando debe pararse.
Este también podrá elegir entre que porcentajes de tiempo trabajará la bomba. Además, la aplicación emitirá una señal de alarma cuando la temperatura interior supere una temperatura excesiva. Todos estos parámetros se encuentran en la parte izquierda y en la parte inferior de la gráfica que se muestra en la figura 24:

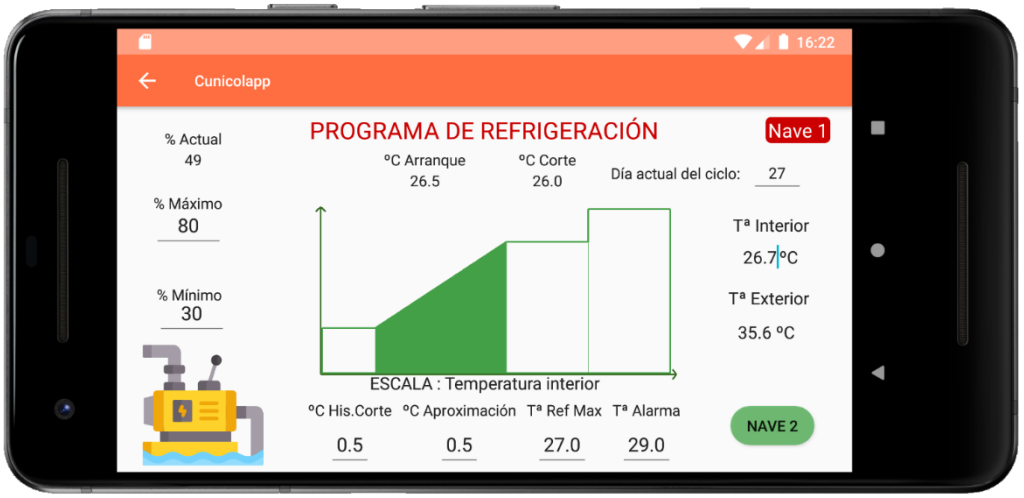


Figura 24: Ventana de Refrigeración 1

El “cooling” de refrigeración trabaja de forma progresiva, es decir, funciona de un mínimo a un máximo de tiempo, para evitar introducir demasiada humedad innecesaria dentro de la nave.

Por lo tanto, la bomba que activa el "cooling" trabaja a diferentes porcentajes dependiendo de la situación en la que se encuentre la nave, es decir, en función de la temperatura interior de la nave y de los parámetros introducidos por el usuario obtendremos un porcentaje u otro. Este porcentaje comienza en un mínimo de tiempo en el cual el "cooling" está en marcha, y aumenta de forma progresiva hasta un porcentaje de 100 % cuando alcanza una temperatura máxima establecida por el usuario. Es decir:

- Cuando la temperatura interior sea igual a la temperatura de arranque, la bomba trabajará al porcentaje mínimo introducido por el usuario.

- Cuando la temperatura interior sea superior a la temperatura de arranque e inferior a la temperatura de referencia máxima, la bomba trabajará en la zona proporcional, cuyo porcentaje se calcula de la siguiente manera:

  Las variables que intervienen en esta operación son las siguientes:

  - %min
  - %max
  - Tª arranque
  - Tª ref.max
  - Tª interior

  Todos estos parámetros son editables por el usuario. La variable Tªinterior correspondería al valor en ºC arrojado por la sonda de temperatura.

  Cálculo de la pendiente de la recta:

$$m = \frac{\%_{max} - \%_{min}}{T^{a}_{ref.max} - T^{a}_{arranque}}$$

  Ecuación de la recta:

$$\%_{actual} = m \cdot \left(T^{a}_{interior} - T^{a}_{arranque}\right) + \%_{min}$$

- En el caso de que la temperatura interior supera a la temperatura de referencia máxima, la bomba comenzara a trabajar al porcentaje máximo introducido por el usuario.

  Si este parámetro es menor a 100 % (como en la figura 24), el porcentaje de trabajo de la bomba alcanzará la potencia máxima, es decir el 100 %, cuando la temperatura interior rebase la temperatura de alarma.

  La gráfica cambiará de forma en función del porcentaje máximo introducido por el usuario, para observar con mayor claridad la zona en la que nos encontramos. Si el porcentaje máximo introducido es el 100 % la gráfica cambiará de forma y se dibujará como se muestra en la figura 25.


Figura 25: Ventana de Refrigeración 2

### 6.2.8. Alimentación

El programa de alimentación se basa en un sistema de encendido y apagado de los silos. A partir de los parámetros que el usuario introduce, el programa es capaz de activar y desactivar los silos en el momento oportuno. El ganadero o ganadera debe introducir la hora de inicio del racionamiento y la duración del mismo, por tanto, si se suman los dos parámetros introducidos, se obtiene la hora en la que los silos deben ser apagados.

Además, el programa controla la cantidad de pienso existente en los diferentes silos. En la explotación existen tres silos con diferentes piensos: el de madres, el de engorde y el de retirada. Solamente podemos proporcionar a los conejos un tipo de pienso, es decir, el programa solo permite tener accionado uno de los silos cuando se raciona el alimento. Para activar cualquiera de los silos basta con seleccionar el *Switch* que tiene asociados cada uno de ellos. Si activamos un nuevo silo, el que estuviera activado se debe desactivar automáticamente.


Figura 26: Ventana de Alimentación

Si la aplicación no fuera un prototipo, se partiría de que tenemos tres sondas las cuales medirían la cantidad de pienso que queda. En este caso, en la aplicación se muestran unos *Scroll bars* (que se observan en la figura 26) los cuales simulan la bajada y la subida del pienso en los diferentes silos cuando estos se encuentran activados. Es decir, si el silo no está accionado el *Scroll bar* no será visible.

A la vez que se desliza el *Scroll bar* hacia la derecha, la cantidad de pienso en el silo seleccionado va disminuyendo.
Cuando el pienso baja de una determinada cantidad, el programa emitirá una señal de alarma para prevenir al ganadero de que debe realizar otro pedido y que tipo de pienso se está acabando. El mensaje que aparecerá será el siguiente.

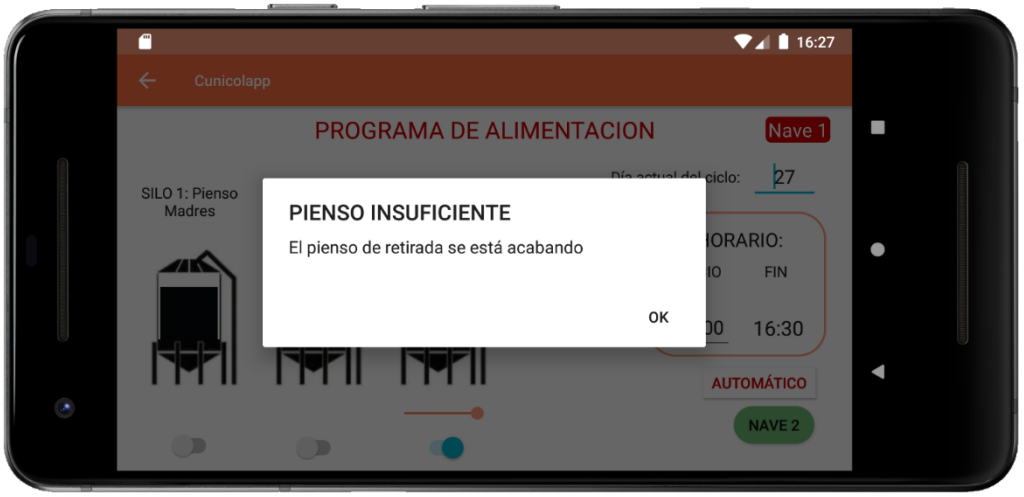


Figura 27: Ventana de Alimentación. Alerta pienso insuficiente.

El programa de alimentación tiene un modo manual y otro automático. En el modo automático el programa comprueba según la hora del sistema si los animales deben ser alimentados o no. En cambio cuando el modo manual está activado, el programa activa el silo que el usuario haya seleccionado sin tener en cuenta la hora que sea.

## 6.2.9. Datos Técnicos de las bandas


Figura 28: Ventana Datos técnicos de las bandas

En la ventana de datos técnicos, se mostrarán todas las bandas creadas por el usuario. Cada vez que el usuario quiera crea una banda debe clicar el botón de agregar banda que aparece en la parte inferior de la figura 28, aparecerá una nueva ventana como la que se observa en la figura 29, donde habrá que rellenar los siguientes datos: la fecha, el número de banda y el estado.
El programa introduce automáticamente los valores de dichos campos utilizando la fecha del día actual, el número de banda consecutivo a la última banda creada y el estado de la banda que por defecto es 1. Estos parámetros pueden ser modificados por el usuario en el momento de crear la banda.


Figura 29: Ventana crear nueva banda

Las bandas se muestran en un *ReciclerView* (figura 28), en el cual podemos deslizar para ver la cantidad total de bandas existentes. El programa nos da la opción de eliminar o editar cualquiera de las bandas existentes. Para editar la banda basta con clicar encima de cualquiera de ellas y el programa te dirigirá directamente a la ventana de editar banda. En cambio, si lo que se quiere es eliminar una banda, el usuario deberá mantener clicada dicha banda hasta que aparezca un mensaje como el de la figura 30. Las bandas se muestran en orden de la más reciente a la más antigua.

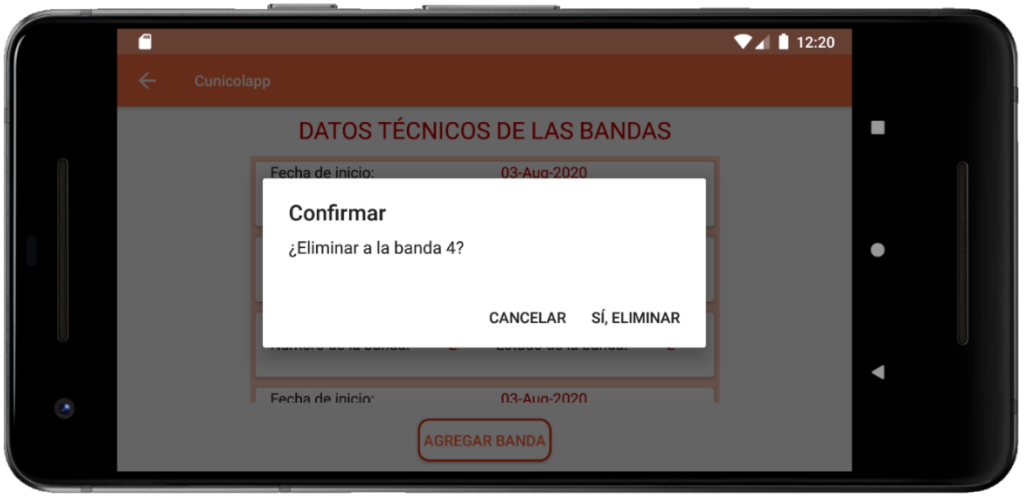


Figura 30: Ventana datos técnicos. Eliminar banda

Al crear una nueva banda el primer parámetro que se debe introducir en la ventana de edición de la banda es el número total de conejas inseminadas, así como el de inseminaciones a conejas nuevas, de 1 a 3 partos, de 4 a 10 partos y de más de 10 partos. Estos datos se encuentran en la parte superior izquierda de la ventana editar banda como se puede observar en la figura 31.

En cuanto a la edición de las bandas, se muestra una nueva ventana en la cual podemos ir añadiendo información sobre cada banda. Esta información se irá rellenando a lo largo de la vida de la banda, hasta que se rellenen los últimos datos, los cuales son el número de conejos y kilos vendidos. Si el usuario introduce dichos parámetros significa que la banda ha llegado a su fin y no podrá ser editada posteriormente.
Sin embargo, sí que tendremos la opción de observar toda la información recogida a lo largo de la vida de cada una de las bandas en la ventana resumen de cada banda.

Por lo tanto, tenemos tres posibles estados en los que se puede encontrar una banda. El estado 1 correspondería con el espacio de tiempo desde que se inseminan las reproductoras, hasta que paren. El estado 2 hace referencia al espacio de tiempo desde que nacen los conejos hasta que se venden, y por último el estado 3 corresponde a la banda finalizada.

Los datos relevantes que se van introduciendo a lo largo de cada ciclo son:

- Número de inseminaciones totales.
- Número de inseminaciones a conejas nuevas.
- Número de inseminaciones de conejas que han tenido de 1 a 3 partos.
- Número de inseminaciones de conejas que han tenido de 4 a 10 partos.
- Número de inseminaciones de conejas que han tenido más de 10 partos.
- Número de palpaciones.
- Número de palpaciones nuevas.
- Número de partos.
- Número de conejos nacidos.
- Número de conejos destetados.
- Número de conejas de reposición.
- Número de conejos que van al engorde.
- Número de bajas en el nido.
- Número de bajas en reproductoras
- Número de bajas en engorde.
- Número de kilos vendidos.
- Número de conejos vendidos.


Figura 31: Ventana editar banda

A la hora de actualizar los diferentes datos de las bandas, se deberá clicar el botón guardar cambios para que estos sean remplazados en la base de datos. Si se han guardado correctamente aparecerá un mensaje en la parte inferior que dirá: “La banda fue actualizada correctamente”.

Con todos estos datos se genera una gran cantidad de información útil tanto para el ganadero o ganadera como para el veterinario. Para acceder a esta información se debe clicar el botón de resumen banda que dirigirá al usuario a la ventana de la figura 32. La información que se muestra es la siguiente:

- Inseminaciones totales.
- Porcentaje de inseminaciones nuevas.
- Porcentaje de inseminaciones de conejas de 1 a 3 partos.

- Porcentaje de inseminaciones de conejas de 4 a 10 partos.
- Porcentaje de inseminaciones de conejas de más de 10 partos.
- Porcentaje de palpaciones positivas.
- Porcentaje de partos sobre inseminaciones.
- Porcentaje de partos sobre palpaciones positivas.
- Nacidos sobre partos.
- Destetados sobre partos.
- Porcentaje de bajas en nidales.
- Porcentaje de inseminaciones nuevas sobre las conejas de reposición.
- Porcentaje de kilos vendidos sobre inseminación.
- Porcentaje de conejos vendidos sobre partos.
- Peso medio de los conejos.


Figura 32: Ventana resumen banda

## 6.2.10. Estadísticas

La ventana de estadísticas sirve para graficar tanto los datos ambientales como los datos técnicos de las bandas.


Figura 33: Ventana de Estadísticas

En la parte izquierda de la pantalla se encuentran los parámetros los cuales obtenemos gracias a las sondas instaladas en la explotación. Estos son: temperatura interior, temperatura exterior, humedad, $CO_2$ y oxígeno.

Al lado de cada nombre se muestra el último dato recogido por la sonda y un botón el cual nos dirigirá a la gráfica del parámetro que queramos consultar. En la siguiente imagen se muestra un ejemplo de la gráfica del dato de temperatura interior:

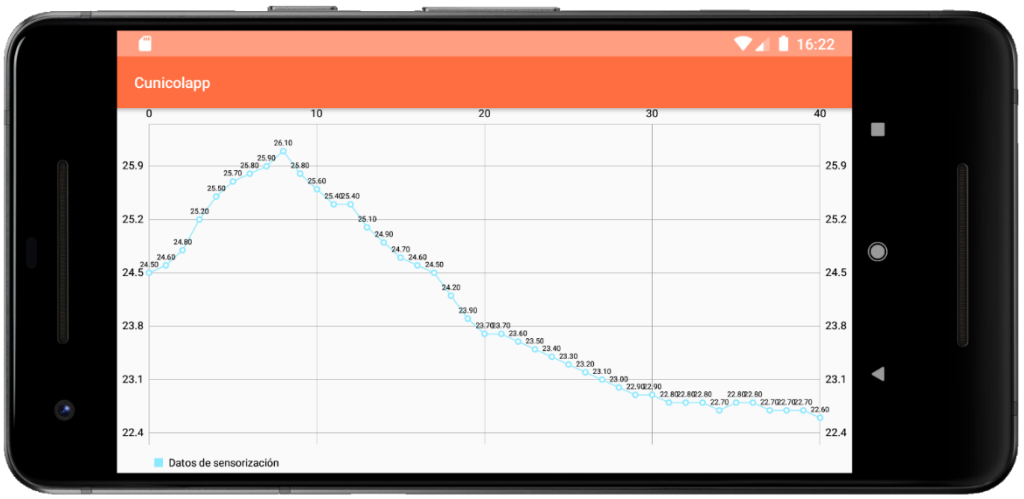


Figura 34: Ventana estadísticas. Datos de sensorización

En cuanto a los datos técnicos de las bandas, se encuentra en la parte derecha de la ventana. El usuario deberá introducir las bandas que quiere comparar, así como los datos técnicos que desea que sean graficados. Como mínimo se deben introducir al menos dos bandas, si se inserta solo una banda el programa no graficará los datos, y saldrá en pantalla un aviso.
Se pueden graficar como máximo 10 bandas.
Cuando el usuario haga clic en la flecha del *Multispinner* aparecerán las bandas y los datos almacenados en la base de datos. A continuación, se muestra cómo se pueden seleccionar los datos.

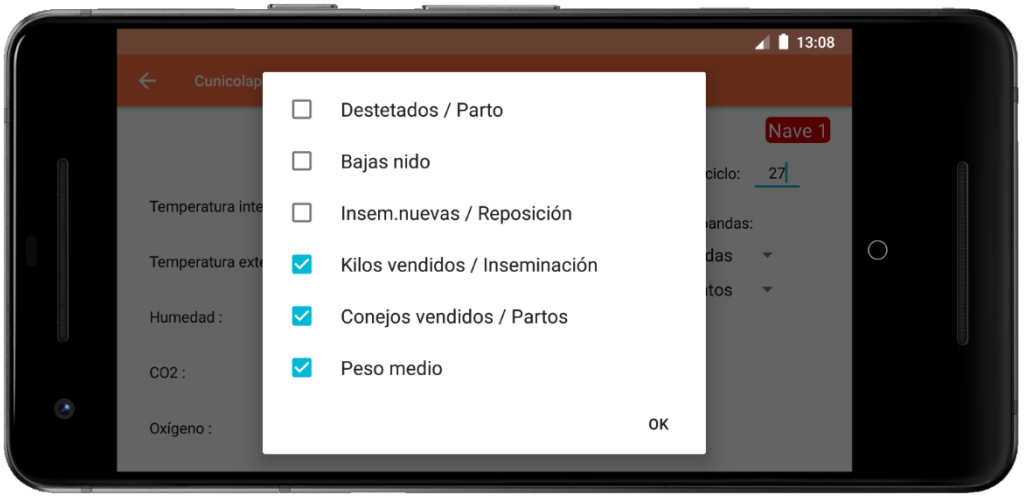


Figura 35: Ventana estadísticas. Selección de datos.

El programa ofrece la opción de graficar los datos de varios modos:

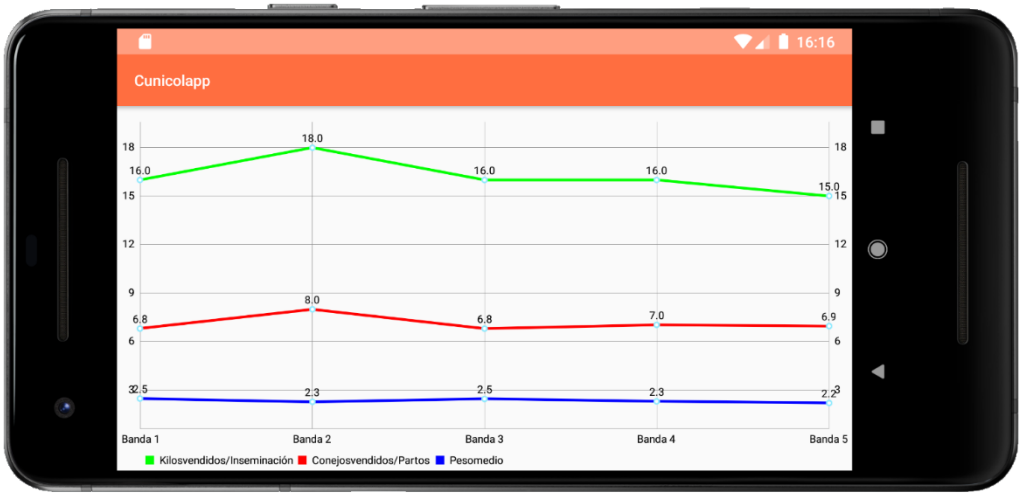


Figura 36: Ventana Estadísticas. Gráfico lineal


Figura 37: Ventana estadísticas. Gráfico de barras

La figura 36 es una gráfica de líneas en la que se representa en el eje x las diferentes bandas y en el eje y los diferentes datos a graficar. En la parte inferior de la gráfica aparece el color de cada dato representado.

La segunda opción que nos ofrece la aplicación (figura 37) es mostrar los datos en un gráfico de barras, lo que ocurre si se selecciona esta opción es que se mostrará en el eje x los datos técnicos de las bandas seleccionados, y se representará con barras de colores diferentes las bandas seleccionadas. De este modo es posible observar los valores de las diferentes bandas para un dato técnico concreto.

## 6.3. Desarrollo de la aplicación

En este apartado se muestra desarrollo del código más importante de la aplicación, comenzando por aspectos generales y seguidamente haciendo referencia al código más importante de cada actividad.

### 6.3.1. Base de datos SQLite

Dado que el presente proyecto es un prototipo de la aplicación se ha optado por utilizar SQLite como base de datos local.

SQLite es una biblioteca que implementa un ligero motor de bases de datos, autónomo, sin servidor y de código abierto, que se caracteriza por mantener el almacenamiento de información persistente de forma sencilla. SQLite es la base de datos más implementada en el mundo con más aplicaciones de las que podemos contar, incluidos varios proyectos de alto perfil.

A diferencia de otros Sistemas gestores de bases de datos como MySQL, SQL Server y Oracle DB, SQLite tiene las siguientes ventajas:

- No requiere el soporte de un servidor: SQLite no ejecuta un proceso para administrar la información, si no que implementa un conjunto de librerías encargadas de la gestión.
- No necesita configuración: libera al programador de todo tipo de configuraciones de puertos, tamaños, ubicaciones, etc.
- Usa un archivo para el esquema: crea un archivo para el esquema completo de una base de datos, lo que permite ahorrarse preocupaciones de seguridad, ya que los datos de las aplicaciones Android no pueden ser accedidos por contextos externos.
- Es de código abierto: está disponible al dominio público de los desarrolladores al igual que sus archivos de compilación e instrucciones de escalabilidad.

Es por eso por lo que SQLite es una tecnología cómoda para los dispositivos móviles. Su simplicidad, rapidez y usabilidad permiten un desarrollo muy amigable.

En la base de datos vamos a almacenar ocho tablas:

- Tabla Bandas
- Tabla Detalles Bandas
- Tabla Pesos
- Tabla temperatura interior
- Tabla temperatura exterior
- Tabla humedad
- Tabla CO2
- Tabla oxígeno

Estas tablas se definen en una clase pública llamada *Utilidades,* que contiene el siguiente código: Se muestran únicamente tres de las ocho tablas.

```
public class Utilidades {
    public static final String NOMBRE_BD="bandas";
    //Constantes campos tabla bandas
    public static final String TABLA_BANDAS="banda";
    public static final String CAMPO_ID="id";
    public static final String CAMPO_FECHA="fecha";
    public static final String CAMPO_NUMERO="numero";
    public static final String CAMPO_ESTADO="estado";

// Constantes campos tabla detalles de las bandas
```

```
    public static final String TABLA_DETALLESBANDAS="detallesBandas";
    public static final String CAMPO_IDDETALLE="iddetalle";
    public static final String CAMPO_INSEMINACIONESTOTALES="inseminacionestotales";
    public static final String CAMPO_INSEMINACIONESNUEVAS="inseminacionesnuevas";
    public static final String CAMPO_INSEMINACIONES1A3PARTOS="inseminaciones1a3partos";
    public static final String CAMPO_INSEMINACIONES4A10PARTOS="inseminaciones4a10partos";
    public static final String CAMPO_INSEMINACIONESMAS10PARTOS="inseminacionesmas10partos";
    public static final String CAMPO_PALPACIONES="palpaciones";
    public static final String CAMPO_PALPACIONESNUEVAS="palpacionesnuevas";
    public static final String CAMPO_PARTOS="partos";
    public static final String CAMPO_NACIDOS="nacidos";
    public static final String CAMPO_DESTETADOS="destetados";
    public static final String CAMPO_REPOSICION="reposicion";
    public static final String CAMPO_BAJASNIDO="bajasnido";
    public static final String CAMPO_ENGORDE="engorde";
    public static final String CAMPO_BAJASREPRODUCTORAS="bajasreproductoras";
    public static final String CAMPO_BAJASENGORDE="bajasengorde";
    public static final String CAMPO_CONEJOSVENDIDOS="vendidos";
    public static final String CAMPO_KILOSVENDIDOS="kilosvendidos";

// Constantes de la tabla peso
    public static final String TABLA_PESO ="kpvivo";
    public static final String CAMPO_IDD = "id";
    public static final String CAMPO_PESO = "peso";


    public static final String CREAR_TABLA_BANDAS= "CREATE TABLE "
            +TABLA_BANDAS + " ("
            +CAMPO_ID+" INTEGER PRIMARY KEY, "
            +CAMPO_FECHA+" INTEGER NOT NULL , "
            +CAMPO_NUMERO+" INTEGER NULL ,"
            +CAMPO_ESTADO+" INTEGER NULL)";


    public static final String CREAR_TABLA_DETALLESBANDAS= "CREATE TABLE "
            +TABLA_DETALLESBANDAS + " ("
            +CAMPO_IDDETALLE + " INTEGER, "
            +CAMPO_INSEMINACIONESTOTALES + " INTEGER NULL , "
            +CAMPO_INSEMINACIONESNUEVAS + " INTEGER NULL , "
            +CAMPO_INSEMINACIONES1A3PARTOS + " INTEGER NULL , "
            +CAMPO_INSEMINACIONES4A10PARTOS + " INTEGER NULL , "
            +CAMPO_INSEMINACIONESMAS10PARTOS + " INTEGER NULL , "
            +CAMPO_PALPACIONES + " INTEGER NULL , "
            +CAMPO_PALPACIONESNUEVAS + " INTEGER NULL , "
            +CAMPO_PARTOS + " INTEGER NULL ,"
            +CAMPO_NACIDOS + " INTEGER NULL ,"
            +CAMPO_DESTETADOS + " INTEGER NULL ,"
            +CAMPO_REPOSICION + " INTEGER NULL , "
            +CAMPO_BAJASNIDO + " INTEGER NULL ,"
            +CAMPO_ENGORDE + " INTEGER NULL ,"
            +CAMPO_BAJASREPRODUCTORAS + " INTEGER NULL ,"
            +CAMPO_BAJASENGORDE + " INTEGER NULL ,"
            +CAMPO_CONEJOSVENDIDOS + " INTEGER NULL ,"
            +CAMPO_KILOSVENDIDOS + " INTEGER NULL)";

    public static final String CREAR_TABLA_PESO = "CREATE TABLE "
            + TABLA_PESO + " ("
            + CAMPO_IDD + " INTEGER, "
            + CAMPO_PESO + " FLOAT)";
}
```

El Android SDK nos provee una serie de clases para administrar nuestro archivo de base de datos en SQLite. La clase que nos permitirá comunicar nuestra aplicación con la base de datos se llama *MyOpenHelper*. Se trata de una clase abstracta que nos provee los mecanismos básicos para la relación entre la aplicación Android y la información.

Para implementar este controlador se debe:

- Crear una clase que extienda de *SQLiteOpenHelper*
- Configurar un constructor apropiado
- Sobrescribir los métodos *onCreate*() y *onUpgrade*()

```
public class MyOpenHelper extends SQLiteOpenHelper{

    public MyOpenHelper(Context context, String name,
SQLiteDatabase.CursorFactory factory, int version) {
        super(context, name, factory, version);
    }
    @Override
    public void onCreate(SQLiteDatabase BaseDeDatos) {
        BaseDeDatos.execSQL(Utilidades.CREAR_TABLA_BANDAS);
        BaseDeDatos.execSQL(Utilidades.CREAR_TABLA_DETALLESBANDAS);
        BaseDeDatos.execSQL(Utilidades.CREAR_TABLA_PESO);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS " + Utilidades.TABLA_BANDAS);
        db.execSQL("DROP TABLE IF EXISTS " + Utilidades.TABLA_DETALLESBANDAS);
        db.execSQL("DROP TABLE IF EXISTS " + Utilidades.TABLA_PESO);
        onCreate(db);
    }
```

A continuación, se muestra la clase *SQLControlador*, que contiene todos los métodos necesarios para escribir, leer, editar y eliminar los datos de las tablas de nuestra base de datos:

- *nuevaBanda*: se crea una nueva banda con fecha, numero de banda y estado de banda.
- *editarBanda* y *editardatosBanda*: para actualizar en la base de datos los parámetros de las bandas y sus detalles.
- *nuevoDetalleBanda*: se crea una nueva tabla donde se almacenan los datos relevantes de las respectivas bandas. Cada componente de la tabla Bandas está relacionado con su componente en la tabla DetallesBandas mediante un id.
- *obtenerBandas* y *obtenerDetallesBanda*: son métodos para la lectura de las diferentes tablas de la base de datos.
- *eliminarBanda*: se elimina tanto la banda como los detalles de la banda seleccionada.

En el siguiente ejemplo solo se mostrará uno de los métodos.

```
public class SQLControlador {
    private MyOpenHelper dbhelper;
    private SQLiteDatabase database;

    public SQLControlador(Context c) {
        dbhelper = new MyOpenHelper(c,NOMBRE_BD,null,1);
    }


    public long nuevaBanda(claseBandas banda) {
        database = dbhelper.getWritableDatabase();
        ContentValues cv = new ContentValues();
        cv.put(Utilidades.CAMPO_FECHA, banda.getFechainicio());
```

```
        cv.put(Utilidades.CAMPO_NUMERO, banda.getNumeroBanda());
        cv.put(Utilidades.CAMPO_ESTADO, banda.getEstadoBanda());

        return database.insert(TABLA_BANDAS, null, cv);
    }
        ...
}
```

La clase Bandas y la clase *DetallesBandas* son las siguientes, ambas incluyen sus respectivos getters and setters:

```
public class claseBandas {
    private int codigo,numeroBanda,estadoBanda;
    private String fechainicio;
    public claseBandas(){}
    public claseBandas(int codigo,String fechainicio, int numeroBanda, int
estadoBanda) {
        this.codigo = codigo;
        this.fechainicio = fechainicio;
        this.numeroBanda = numeroBanda;
        this.estadoBanda = estadoBanda;
    }
...


public class claseDetallesBandas {
    private int id,inseminacionestotales,inseminacionesnuevas,inseminaciones1a3partos,
inseminaciones4a10partos,inseminacionesmas10partos,palpaciones,palpacionesnuevas,
partos,nacidos,inseminacionesmas10partos,palpaciones,palpacionesnuevas,partos,nacidos,
destetados,reposicion,bajasnido,engorde,bajasreproductoras,bajasengorde,
conejosvendidos,kilosvendidos;

    public claseDetallesBandas(){}

    public claseDetallesBandas(int id, int inseminacionestotales, int
     inseminacionesnuevas, int inseminaciones1a3partos, int
     inseminaciones4a10partos, int inseminacionesmas10partos, int palpaciones,
     int palpacionesnuevas, int partos, int nacidos, int destetados, int
     reposicion, int bajasnido, int engorde, int bajasreproductoras, int
     bajasengorde, int conejosvendidos, int kilosvendidos) {
        this.id = id;
        this.inseminacionestotales = inseminacionestotales;
        this.inseminacionesnuevas = inseminacionesnuevas;
        this.inseminaciones1a3partos = inseminaciones1a3partos;
        this.inseminaciones4a10partos = inseminaciones4a10partos;
        this.inseminacionesmas10partos = inseminacionesmas10partos;
        this.palpaciones = palpaciones;
        this.palpacionesnuevas = palpacionesnuevas;
        this.partos = partos;
        this.nacidos = nacidos;
        this.destetados = destetados;
        this.reposicion = reposicion;
        this.bajasnido = bajasnido;
        this.engorde = engorde;
        this.bajasreproductoras = bajasreproductoras;
        this.bajasengorde = bajasengorde;
        this.conejosvendidos = conejosvendidos;
        this.kilosvendidos = kilosvendidos;
    }
...
```

### 6.3.2. Clase Global

Esta clase se ha creado con el fin de almacenar ciertos datos que la aplicación debe conservar de forma permanente. En Android hay varias maneras de almacenar datos: Puedes hacerlo a través de una base de datos local y/o remota, a través de las *SharedPreferences* o combinar ambas formas.

Lo que se busca es que ese valor siga siendo recordado a pesar de cerrar la aplicación o de incluso reiniciar o apagar nuestro móvil.

Las preferencias no son más que datos que una aplicación debe guardar para personalizar la experiencia del usuario, por ejemplo, información personal, opciones de presentación, parámetros de configuración de una aplicación, etc.

Anteriormente hemos utilizado ya uno de los métodos disponibles en la plataforma Android para almacenar datos, como son las bases de datos SQLite. Las preferencias de una aplicación se podrían almacenar por su puesto utilizando este método, y no tendría nada de malo, pero Android proporciona otro método alternativo diseñado específicamente para administrar este tipo de datos: las preferencias compartidas o *SharedPreferences*.
Cada preferencia se almacenará en forma de clave-valor, es decir, cada una de ellas estará compuesta por un identificador único y un valor asociado a dicho identificador. Se pueden almacenar variables de tipo booleano, real, entero y String.  Además, y a diferencia de SQLite, los datos no se guardan en un fichero binario de base de datos, sino en ficheros XML.

La gestión de estas preferencias se centraliza en la clase *SharedPreferences*, que representa una colección de preferencias. A la hora de instanciarlas, estas tienen un identificador, que es la clave o nombre que le hayamos puesto nosotros, y un modo de acceso (en nuestro caso privado).

Para cada ventana de la aplicación tenemos una serie de variables que guardamos en las preferencias compartidas. A continuación, se muestra la clase global con una de las variables, ya que para el resto de las variables se programa de la misma forma.

```
public class claseGlobal extends Application {

    private double temperaturainterior;

    public double getTemperaturainterior() {
        SharedPreferences preferencias = getSharedPreferences("credenciales",
         Context.MODE_PRIVATE);
        String tempinterior = preferencias.getString("tempinterior",String.valueOf(24.9));
        temperaturainterior= Double.parseDouble(tempinterior);
        return temperaturainterior;
    }
    public void setTemperaturainterior(double temperaturainterior) {
        SharedPreferences preferencias = getSharedPreferences("credenciales",
Context.MODE_PRIVATE);
        SharedPreferences.Editor editor = preferencias.edit();
        editor.putString("tempinterior",String.valueOf(temperaturainterior));
        editor.apply();
        this.temperaturainterior = temperaturainterior;
    }
}
...
```

Para acceder a las variables globales desde el archivo .java debemos instanciar la clase Global como se muestra a continuación con el ejemplo de temperatura interior:

```
claseGlobal Global;
Global = (claseGlobal) getApplicationContext();
```

Para modificar el valor de la variable utilizamos la sentencia set:

```
Global.setTemperaturainterior(Double.parseDouble(s.toString()));
```

Para obtener el valor de la variable utilizamos la sentencia get:

```
Global.getTemperaturainterior()
```

### 6.3.3. Actividad

Una aplicación en Android está formada por un conjunto de elementos básicos de interacción con el usuario, conocidos como actividades. Son las actividades las que realmente controlan el ciclo de vida de las aplicaciones, dado que el usuario no cambia de aplicación, sino de actividad. El sistema mantiene una pila con las actividades previamente visualizadas, de forma que el usuario puede regresar a la actividad anterior pulsando la tecla “retorno”.

Una aplicación Android corre dentro de su propio proceso Linux. Este proceso se crea con la aplicación y continuará vivo hasta que ya no sea requerido y el sistema reclame su memoria para asignársela a otra aplicación.

Una característica importante, y poco usual, de Android es que la destrucción de un proceso no es controlada directamente por la aplicación, sino que es el sistema el que determina cuándo destruir el proceso. Lo hace basándose en el conocimiento que tiene de las partes de la aplicación que están corriendo (actividades y servicios), en la importancia de dichas partes para el usuario y en cuánta memoria disponible hay en un determinado momento.

Si tras eliminar el proceso de una aplicación, el usuario vuelve a ella, se crea de nuevo el proceso, pero se habrá perdido el estado que tenía esa aplicación. En estos casos, será responsabilidad del programador almacenar el estado de las actividades, si queremos que cuando sean reiniciadas conserven su estado.

Como vemos, Android es sensible al ciclo de vida de una actividad; por lo tanto, necesitas comprender y manejar los eventos relacionados con el ciclo de vida si se quiere crear aplicaciones estables.

Una actividad en Android puede estar en uno de estos cuatro estados:

- Activa (Running): La actividad está encima de la pila, lo que quiere decir que es visible y tiene el foco.
- Visible (Paused): La actividad es visible pero no tiene el foco. Se alcanza este estado cuando pasa a activa otra actividad con alguna parte transparente o que no ocupa toda la pantalla. Cuando una actividad está tapada por completo, pasa a estar parada.
- Parada (Stopped): Cuando la actividad no es visible. El programador debe guardar el estado de la interfaz de usuario, preferencias, etc.
- Destruida (Destroyed): Cuando la actividad termina al invocarse el método finish(), o es matada por el sistema.

Todas las clases tipo *activity* tienen estos ciclos de vida, y es conveniente conocerlos para poder programar correctamente algunos métodos. Cada vez que una actividad cambia de estado se van a generar eventos que podrán ser capturados por ciertos métodos de la actividad. A continuación,

se muestra un esquema que ilustra los métodos que capturan estos eventos y se procede a explicar cada uno de ellos.

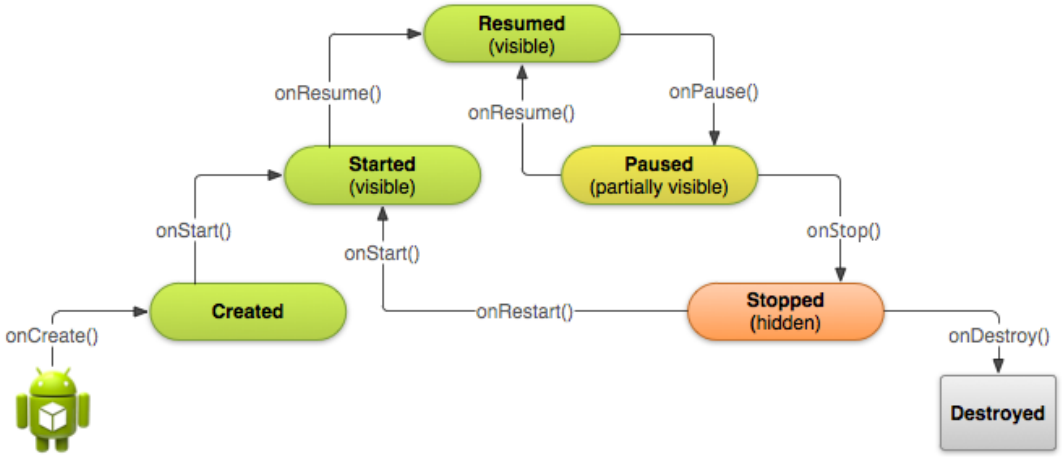


Figura 38: Ciclo de vida de un actividad (*Pausar y Reanudar una Actividad - Gestionar el Ciclo de Vida de una Actividad*, n.d.)

- onCreate(Bundle): Se llama en la creación de la actividad. Se utiliza para realizar todo tipo de inicializaciones, como la creación de la interfaz de usuario o la inicialización de estructuras de datos. Puede recibir información de estado de la actividad (en una instancia de la clase Bundle), por si se reanuda desde una actividad que ha sido destruida y vuelta a crear.
- onStart(): Nos indica que la actividad está a punto de ser mostrada al usuario.
- onResume(): Se llama cuando la actividad va a comenzar a interactuar con el usuario. Es un buen lugar para lanzar las animaciones y la música.
- onPause(): Indica que la actividad está a punto de ser lanzada a segundo plano, normalmente porque otra actividad es lanzada. Es el lugar adecuado para detener animaciones, música o almacenar los datos que estaban en edición.
- onStop(): La actividad ya no va a ser visible para el usuario. Hay que tener en cuenta que si hay muy poca memoria es posible que la actividad se destruya sin llamar a este método.
- onRestart(): Indica que la actividad va a volver a ser representada después de haber pasado por onStop().
- onDestroy(): Se llama antes de que la actividad sea totalmente destruida. Por ejemplo, cuando el usuario pulsa el botón de volver o cuando se llama al método finish(). Ojo si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.

Cada vez que se accede a cualquiera de las pantallas es necesario que las variables conserven el mismo valor que tenían la última vez, o que se actualicen si algún parámetro ha cambiado. Para ello se ha creado en todas las ventanas la función *CargarDatos*, que lo que hace es acceder a las preferencias compartidas y cargar el valor de cada variable en su respectivo *TextView*, *editText*… Dicha función se ubica en el método *OnResume* por razones expuestas con anterioridad al comprender como funciona el ciclo de vida de una actividad.

```
protected void onResume (){
    super.onResume();
    cargardatos();
}
```

### 6.3.4. Ventana principal

La ventana principal se compone de un *Grid Layout* que permite dividir la pantalla en ocho partes iguales, en ocho *CardViews* en las que dentro se encuentra el nombre de la ventana a acceder, un icono representativo y un *ToolBar* como se observa en la figura 11 de la página 39.
En cuanto al código más importante en el archivo .Java sería las acciones que debe realizar la aplicación al seleccionar los *ToolBars* de las diferentes *Cardviews*. En este caso se muestra de nuevo el código del estado general, que redirigirá al usuario al estado general de la nave 1 o de la nave 2.

```
toolbarCardEG.inflateMenu(R.menu.menucard);
toolbarCardEG.setOnMenuItemClickListener(new Toolbar.OnMenuItemClickListener()
{
    @Override
    public boolean onMenuItemClick(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.action_option1:
                Intent intent = new Intent(ventanaprincipal.this,
                        ventanaestadogeneral.class);
                startActivity(intent);
                break;
            case R.id.action_option2:
                Intent intent2 = new Intent(ventanaprincipal.this,
                        ventanaestadogeneral2.class);
                startActivity(intent2);
        }
        return true;
    }
});
```

En primer lugar, se infla el menú en el *toolbar*. El archivo *menucard*, que se encuentra en la carpeta menu, dentro del directorio res, es el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context=".ui.MainActivity">

    <item
        android:id="@+id/action_option1"
        android:title="Nave 1"
        app:showAsAction="never" />
    <item
        android:id="@+id/action_option2"
        android:title="Nave 2"
        app:showAsAction="never" />
</menu>
```

Además, puesto que la pantalla del dispositivo puede tener un tamaño relativamente reducido, se ha optado por habilitar el *CardView* completo para hacer clic en él, además de poder acceder clicando el *ToolBar*. Es decir, con seleccionar cualquier parte del cuadrado la aplicación nos mostrará las opciones de Nave 1 o Nave 2. Para lograr este cometido basta con añadir la siguiente línea de código.

```
estadogeneral.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        toolbarCardEG.showOverflowMenu();
    }
});
```

### 6.3.5. Ventana estado general

La parte más importante de la ventana estado general, es cargar los datos de los sensores y actuadores existentes en la explotación. Esta pantalla permite al usuario ver que está ocurriendo en cada nave, por lo tanto, una vez que se entra en ella lo primero que se hace es comprobar si los actuadores están activos o no. Para ello se consultan las preferencias compartidas y se muestra la imagen correspondiente para cada actuador en función de si está encendido o apagado.

```
public void cargardatos(){

 temperaturainterior.setText(String.valueOf(Global.getTemperaturainterior()));
 grupoactual.setText(String.valueOf(Global.getGrupo()));
 qactual.setText(String.valueOf(Global.getQactual()));
 porcentajeactualventilacion.setText(String.valueOf(Global.getPorcentajevactual());
 diaactualciclo.setText(String.valueOf(Global.getDiaactualdelciclo1()));

    //COMPROBAR FUNCIONAMIENTO DEL COOLING
    if(Global.isCooling()){
        //Encender el cooling
        coolingOn.setVisibility(View.VISIBLE);
        coolingOff.setVisibility(View.INVISIBLE);
    }else{
        coolingOn.setVisibility(View.INVISIBLE);
        coolingOff.setVisibility(View.VISIBLE);
    }
    //COMPROBAR SI LOS VENTILADORES ESTAN ENCENDIDOS
    if(Global.getPorcentajevactual()!=0){
        switch(Global.getGrupo()){
            case 1:
                ventiladorg1Off.setVisibility(View.INVISIBLE);
                ventiladorg1On.setVisibility(View.VISIBLE);
                break;
            case 2:
                ventiladorg1Off.setVisibility(View.INVISIBLE);
                ventiladorg1On.setVisibility(View.VISIBLE);
                ventiladorg2Off.setVisibility(View.INVISIBLE);
                ventiladorg2On.setVisibility(View.VISIBLE);
                break;
            case 3:
                ventiladorg1Off.setVisibility(View.INVISIBLE);
                ventiladorg1On.setVisibility(View.VISIBLE);
                ventiladorg2Off.setVisibility(View.INVISIBLE);
                ventiladorg2On.setVisibility(View.VISIBLE);
                ventiladorg3Off.setVisibility(View.INVISIBLE);
                ventiladorg3On.setVisibility(View.VISIBLE);
                break;
            case 4:
                ventiladorg1Off.setVisibility(View.INVISIBLE);
                ventiladorg1On.setVisibility(View.VISIBLE);
                ventiladorg2Off.setVisibility(View.INVISIBLE);
                ventiladorg2On.setVisibility(View.VISIBLE);
                ventiladorg3Off.setVisibility(View.INVISIBLE);
                ventiladorg3On.setVisibility(View.VISIBLE);
                ventiladorg4Off.setVisibility(View.INVISIBLE);
                ventiladorg4On.setVisibility(View.VISIBLE);
                break;
        }
        if (Global.isGrupoapoyo()){
            //debe encenderse el de apoyo y los que ya estén

            ventiladorgapoyOff.setVisibility(View.INVISIBLE);
```

```
            ventiladorgapoyOn.setVisibility(View.VISIBLE);
        }
    }
    //COMPRUEBO SI LAS LUCES ESTAN ENCENDIDAS
    if (Global.isLuz()){
        iluminacionOn.setVisibility(View.VISIBLE);
        iluminacionOff.setVisibility(View.INVISIBLE);
    }else{
        iluminacionOn.setVisibility(View.INVISIBLE);
        iluminacionOff.setVisibility(View.VISIBLE);
    }
    //COMPRUEBO CALEFACCION
    if (Global.isCalefaccion()){
        canonOn.setVisibility(View.VISIBLE);
        canonOff.setVisibility(View.INVISIBLE);
    }else{
        canonOn.setVisibility(View.INVISIBLE);
        canonOff.setVisibility(View.VISIBLE);
    }
    //COMPRUEBO SILOS
    if (Global.isSilos()){
        silosOn.setVisibility(View.VISIBLE);
        silosOff.setVisibility(View.INVISIBLE);
    }else{
        silosOn.setVisibility(View.INVISIBLE);
        silosOff.setVisibility(View.VISIBLE);
    }
}
```

Además, como hay tres parámetros en la pantalla que son editables (día del ciclo, temperatura interior, temperatura exterior), se programa un *AddTextListener* para que se almacenen en las preferencias compartidas el valor actualizado por el usuario para cada parámetro. A continuación, se muestra el ejemplo de la temperatura interior:

```
temperaturainterior.addTextChangedListener(new TextWatcher() {
    @Override
    public void beforeTextChanged(CharSequence s, int start, int count, int
after) {
    }
    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count)
{
    }
    @Override
    public void afterTextChanged(Editable s) {
        Global.setTemperaturainterior(Double.parseDouble(s.toString()));
    }
});
```

Por último, se han implementado funciones de acceso para las diferentes ventanas que tiene la aplicación. El nombre de cada función corresponde con el método *OnClick* de las respectivas ventanas. La función de acceso a la ventana iluminación es la siguiente:

```
public void accesoiluminacion(View view) {
    Intent intent = new Intent(this, ventanailuminacion.class);
    startActivity(intent);
}
```

### 6.3.6. Ventana iluminación

La ventana de iluminación como se ha explicado anteriormente tiene una gráfica donde el usuario puede comprobar de un solo vistazo cuanto tiempo de luz van a tener las conejas. Además, con el icono de la bombilla se sabe si la iluminación de la nave está encendida o apagada en ese momento.
En primer lugar, cuando se accede a la ventana de iluminación, se cargan los datos de las variables en pantalla del mismo modo que se hace en la ventana de estado general.

En cuanto a la gráfica, se ha utilizado la biblioteca *Canvas* para la dibujar y colorear las diferentes partes de ésta. A continuación, se muestra el código utilizado:

```
    static class Lienzo extends View {

    private int diaactualdelciclo=0;
    private int diainicioflushing=11;
    private int diafinflushing=18;
    private int diafinrampa=21;
    private int diainiciosinluz=77;
    private int diafinsinluz=84;
    private int diafinciclo=84;

public Lienzo(Context context) {
    super(context);
}
protected void onDraw(Canvas canvas) {
    //canvas.drawRGB(255, 255, 0);
    @SuppressLint("DrawAllocation")
    Paint pincel1 = new Paint();
    pincel1.setARGB(255, 255, 0, 0);
    pincel1.setColor(getResources().getColor(R.color.md_green_600));

    //dibujo grafica
    canvas.drawLine(420, 490, 420, 640, pincel1);
    canvas.drawLine(420, 370, 420, 640, pincel1);
    canvas.drawLine(595, 370, 595, 640, pincel1);
    canvas.drawLine(715, 490, 715, 640, pincel1);
    canvas.drawLine(1040, 490, 1040, 640, pincel1);
    canvas.drawLine(1145, 490, 1145, 640, pincel1);

    pincel1.setStrokeWidth(4);
    canvas.drawLine(204, 640, 1250, 640, pincel1);
    canvas.drawLine(204, 640, 204, 490, pincel1);
    canvas.drawLine(204, 490, 420, 490, pincel1);
    canvas.drawLine(420, 490, 420, 370, pincel1);
    canvas.drawLine(420, 370, 595, 370, pincel1);
    canvas.drawLine(595, 370, 715, 490, pincel1);
    canvas.drawLine(715, 490, 1250, 490, pincel1);
    canvas.drawLine(1250, 490, 1250, 640, pincel1);

    //dibujoflecha
    pincel1.setStrokeWidth(3);
    canvas.drawLine(204, 665, 1200, 665, pincel1);
    canvas.drawLine(204, 665, 204, 740, pincel1);
    canvas.drawLine(204, 740, 1200, 740, pincel1);
    canvas.drawLine(1200, 740, 1200, 760, pincel1);
    canvas.drawLine(1200, 760, 1250, 702, pincel1);

    canvas.drawLine(1200, 665, 1200, 645, pincel1);
    canvas.drawLine(1200, 645, 1250, 702, pincel1);
```

```
if (diaactualdelciclo<diainicioflushing){
    pincel1.setColor(getResources().getColor(R.color.md_green_600));
    canvas.drawRect(204, 490, 420, 640, pincel1);
}
if (diaactualdelciclo>=diainicioflushing &&
    diaactualdelciclo<diafinflushing){
    pincel1.setColor(getResources().getColor(R.color.md_green_600));
    canvas.drawRect(420, 370, 595, 640, pincel1);
    }
if (diaactualdelciclo>=diafinflushing && diaactualdelciclo<diafinrampa){
    pincel1.setColor(getResources().getColor(R.color.md_green_600));
    pincel1.setStyle(Paint.Style.FILL);
    canvas.drawRect(595, 490, 715, 640, pincel1);
    Path triangulo = new Path();
    triangulo.moveTo(595, 490);
    triangulo.lineTo(595, 370);
    triangulo.lineTo(715, 490);
    canvas.drawPath(triangulo, pincel1);
}
if (diaactualdelciclo>=diafinrampa && diaactualdelciclo<diainiciosinluz){
    pincel1.setColor(getResources().getColor(R.color.md_green_600));
    canvas.drawRect(715, 490, 1040, 640, pincel1);
    }
if (diaactualdelciclo>=diainiciosinluz && diaactualdelciclo<diafinsinluz){
    pincel1.setColor(getResources().getColor(R.color.md_green_600));
    canvas.drawRect(1040, 490, 1145, 640, pincel1);
    }
if (diaactualdelciclo>=diafinsinluz && diaactualdelciclo<=diafinciclo){
    pincel1.setColor(getResources().getColor(R.color.md_green_600));
    canvas.drawRect(1145, 490, 1250, 640, pincel1);
    }
}
public void diaactual (int dia){
    diaactualdelciclo=dia;
    this.invalidate();
}
public void diainicioflushing (int dia){
    diainicioflushing=dia;
    this.invalidate();
}

public void diafinflushing(int dia) {
    diafinflushing=dia;
    this.invalidate();
}
public void diainiciosinluz (int dia){
    diainiciosinluz=dia;
    this.invalidate();
}
public void diafinrampa (int dia){
    diafinrampa=dia;
    this.invalidate();
}
public void diafinsinluz (int dia){
    diafinsinluz=dia;
    this.invalidate();
}
public void diafinciclo (int dia){
    diafinciclo=dia;
    this.invalidate();
```

```
    }
}
```

Para utilizar esta clase y mostrarla en el archivo .XML debemos añadir las siguientes líneas:

```
ventanailuminacion.Lienzo graficailuminacion;

graficailuminacion = new Lienzo(this);
layoutiluminacion.addView(graficailuminacion);
```

La primera parte de código dentro del método *OnDraw* corresponde a las líneas de código programadas para dibujar la gráfica y la flecha donde se muestran los diferentes días (parámetros editables). Las sentencias condicionales que van a continuación son para dibujar de color verde la zona donde se encuentra la nave según su día de ciclo. Es decir, si algún parámetro como el día del ciclo o cualquiera de los parámetros que se encuentra dentro de la flecha son modificados por el usuario, se llama a las funciones que se encuentran al final de la clase Lienzo, y lo que ocurre es que se actualiza el valor de la variable modificada y se redibuja la gráfica si es necesario.

Por lo tanto, todos los parámetros editables tienen asociados un AddTextListener como en el siguiente ejemplo, donde se actualiza la variable en las preferencias compartidas y se llama a la función correspondiente dentro de la graficailuminación:

```
diaactual.addTextChangedListener(new TextWatcher() {
     @Override
     public void beforeTextChanged(CharSequence s, int start, int count, int
after) {
     }
     @Override
     public void onTextChanged(CharSequence s, int start, int before, int
count) {
     }
     @Override
     public void afterTextChanged(Editable s) {
         try{
             Global.setDiaactualdelciclo1(Integer.parseInt(s.toString()));
             graficailuminacion.diaactual(Integer.parseInt(s.toString()));
            }
         catch (Exception e){
         }
     }
 });
```

La función *comprobariluminacion*() que se llama al cargar los datos, es la que comprueba si con los valores de duración e inicio que ha introducido el usuario y en función de la hora del sistema, la luz de la explotación debería estar encendida o apagada. Es decir, en primer lugar, se obtiene de las preferencias los datos a introducir por el usuario y se comparan con la hora real del sistema, si la luz debe estar encendida, se muestra la imagen de la bombilla a color y si debe estar apagada la imagen se muestra en blanco y negro. Además, se guarda en una variable booleana si la luz está encendida o apagada. En las primeras líneas de código se utiliza la función *parsear()*, dicha función se ha creado para separar las horas y los minutos de la variable *String* duración, hora de inicio y hora de fin, con el objetivo de que sea más fácil hacer las operaciones de comparación.

```
public void comprobariluminacion(){
    h2= parsear(Global.getDuracion())[0];
    m2= parsear(Global.getDuracion())[1];
    h1 = parsear(Global.getHoraInicio())[0];
    m1 = parsear(Global.getHoraInicio())[1];
```

```
    h = parsear(Global.getHoraFin())[0];
    m = parsear(Global.getHoraFin())[1];

    java.util.Calendar c = java.util.Calendar.getInstance();
    hora = c.get(java.util.Calendar.HOUR_OF_DAY);
    minuto = c.get(java.util.Calendar.MINUTE);
    segundo = c.get(java.util.Calendar.SECOND);

    if ((hora >= h1)&&(hora <h))
    {
        imagenbombillaencendida.setVisibility(View.VISIBLE);
        imagenbombillaapagada.setVisibility(View.INVISIBLE);
        Global.setLuz(true);
    } else
    {
        imagenbombillaencendida.setVisibility(View.INVISIBLE);
        imagenbombillaapagada.setVisibility(View.VISIBLE);
        Global.setLuz(false);
    } if ((hora ==h)&&(minuto<m)){
        imagenbombillaencendida.setVisibility(View.VISIBLE);
        imagenbombillaapagada.setVisibility(View.INVISIBLE);
        Global.setLuz(true);
    }
}
```

Para saber en qué modo (manual o automático) se encuentra el usuario se ha creado en las preferencias una variable de tipo booleano encargada de esta tarea. Cada vez que el *ToggleButton* cambia de posición ocurre lo siguiente:

```
modo.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked)
{
        if (isChecked) {
            // MODO AUTOMATICO
            comprobariluminacion();
            Global.setModoIluminacion(true);
        } else {
            // MODO MANUAL
            Global.setModoIluminacion(false);
        }
    }
});
```

El usuario debe seleccionar la duración, la duración de “flushing” y la hora de inicio para modificarla. Para asegurarnos de que el usuario introduce correctamente los datos se ha implementado un *TimePickerDialog*. Esta función es exactamente igual para los tres parámetros. A continuación, se muestra la de modificar la hora de inicio:

```
public void cambiarhorainicio (View view){

    final Calendar cldr = Calendar.getInstance();
    final int hora = cldr.get(Calendar.HOUR_OF_DAY);
    final int minuto = cldr.get(Calendar.MINUTE);
    //no se si es final o no
    TimePickerDialog recogerhorainicio = new
TimePickerDialog(ventanailuminacion.this, new
TimePickerDialog.OnTimeSetListener()
    {@Override
    public void onTimeSet(TimePicker timePicker, int horaDia, int minutos) {
        h1=horaDia;
```

```
        m1=minutos;
        String horaF =(horaDia < 10)? "0" + horaDia : String.valueOf(horaDia);
        String minutoF =(minutos < 10)? "0" + minutos :
String.valueOf(minutos);
        String AM_PM = (horaDia < 12 ) ? "a.m.":"p.m.";
        inicio.setText(horaF + ":" + minutoF + " " + AM_PM);
        Global.setHoraInicio(horaF + ":" + minutoF);
        calculohorafin();
        comprobariluminacion();
     }
    }, 0, 0, false);
    recogerhorainicio.show();
}
```

En la función anterior, tras recoger la nueva hora de inicio, esta se muestra en su respectivo *EditText,* que en este caso se llama *inicio.* Seguidamente se guarda en las preferencias compartidas la nueva hora de inicio, y se calcula cual será la nueva hora de fin con la llamada a la función *calculohorafin(),* por último se comprueba de nuevo la iluminación para encender o apagar la bombilla.

La función *calculohorafin()* es la encargada de obtener las variables necesarias de las preferencias y para realizar el cálculo de la suma entre la duración y la hora de inicio, seguidamente se muestra en pantalla la hora final y se actualiza en las preferencias.

### 6.3.7. Ventana ventilación

Este programa se encarga de calcular los kilos de peso vivos que hay en cada momento en la explotación, para ventilar la nave en función de este parámetro. Para ello hemos almacenado en la base de datos cual es el peso estimado de los conejos cada día de crecimiento.

El programa de ventilación comienza cuando los conejos son destetados de las madres y se trasladan a estas a una nave vacía donde estarán solas durante una semana (día 0 del ciclo). Las conejas en ese momento estarán preñadas, por lo que a los 7 días parirán los conejos de una nueva banda.
En la base de datos donde se almacena lo que pesa cada conejo, para los días del 0 al 6 los gazapos no pesarán nada puesto que la coneja todavía no habrá parido. A partir del día 7 hasta el día 77 los gazapos irán creciendo hasta alcanzar el peso esperado.

Por lo tanto, cada vez que cambie alguno de estos parámetros: el día del ciclo, las madres, camadas y número de gazapos por camada que haya en la nave, se calcularán los kilos de peso vivo llamando a la función *calcularkpv()* que es la encargada de esta tarea.

```
public void calcularkpv(){
    MyOpenHelper admin = new MyOpenHelper(this, NOMBRE_BD ,null, 1);
    SQLiteDatabase BaseDeDatabase = admin.getWritableDatabase();

    String diaactual = diaactualciclo.getText().toString();
    int numerodegazapos= Integer.parseInt(gazapos.getText().toString());
    int numerodemadres= Integer.parseInt(madres2.getText().toString());

    if(!diaactual.isEmpty() /*&& numerodegazapos!=0 && numerodemadres!=0*/){
        @SuppressLint("Recycle") Cursor fila = BaseDeDatabase.rawQuery
                ("select peso from kpvivo where id =" + diaactual, null);

        if(fila.moveToFirst()){

            double pesogazapos = Double.parseDouble(fila.getString(0));
            double pesogazapostotal = (numerodegazapos*pesogazapos);
```

```
            int pesomadrestotal = (numerodemadres*4000);
            double pesokv= (pesogazapostotal+pesomadrestotal);
            int pesokvg =(int)pesokv;
            int pesokvv =pesokvg/1000;
            String resultadopesokilovivo = String.valueOf(pesokvv);
            kpv.setText(resultadopesokilovivo);
            calcularq();
            calculorationventilacion();

            BaseDeDatabase.close();
        } else {
            Toast.makeText(this,"No existe el dia", Toast.LENGTH_SHORT).show();
            BaseDeDatabase.close();
        }

    } else {
        Toast.makeText(this, "Debes introducir el dia del ciclo actual",
Toast.LENGTH_SHORT).show();
    }
}
```

Una vez obtenida la cantidad de kilos de peso vivo, se procede a calcular la ventilación mínima y máxima para ver qué grupo de ventilación será necesario activar. En el apartado 6.2.5. se ha mencionado como se realiza este cálculo. La función encargada de realizar esta tarea se llama *calcularq()* y es llamada dentro de la función *calcularkpv()* y cada vez que algunos de los ratios son modificados.
Si cuando se calcule la ventilación máxima se supera los metros cúbicos del grupo 4, se establecerá dicho grupo como grupo a activar y su capacidad como qmax.
Seguidamente se llama a la función *calculorationventilacion()*, que es la encargada de calcular los $m^3/h$ de ventilación necesaria y el porcentaje de ventilación actual. Esta función mostrará en pantalla los datos calculados y a su vez serán almacenados en las preferencias. Esta función también se invoca cada vez que se ejecuta la función *calcularq()* y además cuando se modifica el valor de alguna de las tres temperaturas .

En esta función también se ha programado las alarmas de cooling y grupo de apoyo, ya que éstas se activarán en función de la temperatura de referencia que se obtenga. El cooling se encenderá cuando la temperatura interior supere en 1.5 grados el valor de temperatura máxima introducido por el usuario.
El grupo de apoyo estará activo cuando la temperatura interior sea igual o superior a la temperatura mínima establecida por el usuario en la pantalla de grupos.
Cuando se activan cualquiera de estas alarmas lo que aparece en pantalla es un *AlertDialog*, que informa al usuario de lo que está ocurriendo y además le permite acceder a la pantalla correspondiente. Se muestra uno de los códigos como ejemplo:

```
public void mostraralertacooling(){
new AlertDialog.Builder(this)
    .setCancelable(true)
    .setTitle("EL COOLING HA SIDO ENCENDIDO")
    .setMessage("La temperatura de la nave ha ascendido a
                "+Global.getTemperaturainterior()+" ºC")
    .setPositiveButton("Acceso Refrigeración", new
DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
           Intent intent = new Intent(ventanaventilacion.this,
               ventanarefrigeracion.class);
           startActivity(intent);
       }
```

```
    })
    .setNegativeButton(android.R.string.no, new
DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            dialog.dismiss();
        }
    })
    .show();
```

Como ya se ha comentado anteriormente desde esta ventana se puede acceder a la ventana grupos. En la parte derecha se puede modificar cualquiera de los valores de $m^3/h$ de cada uno de los grupos. El código desarrollado permite almacenar el nuevo valor en preferencias, así como mostrarlo en pantalla.
Para la gráfica que se muestra se ha utilizado el mismo método explicado anteriormente en el apartado 6.3.6. Así mismo se han desarrollado las funciones necesarias para llamar a la clase desde fuera, lo que nos permite modificar la parte de la gráfica que se debe colorear según en qué zona se encuentre la explotación. También se incluye una función en la gráfica que informa si el porcentaje máximo introducido por el usuario es 100 o menor. En el caso de que sea menor, la gráfica será redibujada dividiéndose en cuatro secciones en lugar de en tres.

Por último, la pantalla de ventilación mínima simplemente presenta los datos de forma más visual, lo que puede ser de ayuda para el usuario al solo tener que echar un vistazo para comprender lo que está ocurriendo en la nave.
Para el dibujo que aparece en el medio de la ventana se ha utilizado de nuevo la biblioteca *Canvas*. Esta pantalla se basa en una interfaz de visualización, los únicos parámetros que se pueden modificar son la temperatura mínima y la máxima.

### 6.3.8. Ventana calefacción

Como en el resto de actividades al comienzo de esta ventana se ejecuta la función *cargardatos().*
Esta ventana se ha dibujado una figura utilizando la librería *Canvas* ya explicada anteriormente, así como su declaración e implementación.
De la misma manera se utiliza el método *AddTextListener* para modificar en las preferencias compartidas aquellos parámetros que son editables.
Además cada vez que varía alguna temperatura o los grados de histéresis se ejecuta la función *ComprobarestadoCalefacción()*, la cual activa o desactiva la imagen del cañón de propano en la pantalla del usuario, así como actualizar la variable booleana creada para la calefacción en las preferencias compartidas.

### 6.3.9. Ventana refrigeración

El programa de refrigeración funciona de forma muy parecida al grupo de ventilación de apoyo. La gráfica que se dibuja en pantalla es la misma y se utiliza el mismo método. Sin embargo, lo único que cambia son el nombre de las funciones, que ahora se sustituyen por las variables de refrigeración, y las sentencias condicionales *if* que se muestran a continuación y según las cuales estaremos en una zona de la gráfica u en otra.

- `if (tint <= Tarranque && tint > tcorte)`
- `if (tint > Tarranque && tint < Trefmax)`
- `if(tint >= Trefmax && tint < Talarma)`
  `//la gráfica se compone de 4 zonas`
  `if (porcenmaxacien)`
  `//la gráfica se compone de 3 zonas`
- `if(tint >= Talarma)`
- `if(porcenmaxacien)`

Podemos distinguir dos funciones principales: *actualizarcambios()* y *Calculoporcentaje().*
La primera sirve para que cada vez que se modifica un dato importante que afecta al estado de la gráfica o a los valores de arranque y corte, se recalculen los valores, se muestren en pantalla y se guarden en las preferencias compartidas.
La segunda función calcula el porcentaje de refrigeración según la temperatura interior. Si el porcentaje es cero, el "cooling" estará apagado. Si la temperatura interior es superior a la temperatura de alarma, aparecerá un mensaje de aviso en el que pondrá que la temperatura es demasiado elevada. Una vez calculado, guardado y mostrado el porcentaje, se procede a comprobar con la función *cooling()* si la bomba estará encendida o apagada y por tanto debe mostrarse o no la imagen a color.

```
public void Calculoporcentaje() {
    int poractual;
    double tempint = Double.parseDouble(Tempint.getText().toString());
    double temparranque = Double.parseDouble(tarranque.getText().toString());
    double temprefmax = Double.parseDouble(trefmax.getText().toString());
    double tempalarma = Double.parseDouble(talarma.getText().toString());
    int pormin = Integer.parseInt(porcentajemin.getText().toString());
    int pormax = Integer.parseInt(porcentajemax.getText().toString());
    try {
        if (tempint >= temparranque && tempint <= temprefmax) {
            double m = ((pormax - pormin) / (temprefmax - temparranque));
            poractual = (int) ((m * (tempint - temparranque)) + pormin);
            porcentajeactual.setText(String.valueOf(poractual));
        } else if (tempint > temprefmax && tempint< tempalarma) {
            poractual=pormax;
            porcentajeactual.setText(String.valueOf(poractual));
        } else if (tempint >= tempalarma) {
            poractual=100;
            porcentajeactual.setText(String.valueOf(poractual));
            Toast.makeText(this, "TEMPERATURA DEMASIADO ELEVADA",
Toast.LENGTH_LONG).show();
        } else {
            poractual=0;
            porcentajeactual.setText(String.valueOf(poractual));
        }
        Global.setPorcentajeractual(poractual);

    } catch (Exception e) {}
    if(Integer.parseInt(porcentajeactual.getText().toString())!=0){
        Global.setCooling(true);
    }else{Global.setCooling(false);
    }
    cooling();
}
```

## 6.3.10. Ventana alimentación

Al acceder al programa de alimentación debemos saber cuál de los tres silos estaba activado y cuál era el modo, para ello se ha programado una variable en las preferencias compartidas donde se guarda el estado del silo y el modo de alimentación. En la función cargar datos se comprueban estas dos variables, se cargan en pantalla los datos y se llama a la función *comprobarsilos().*
Esta función, que se muestra a continuación, se encarga de comprobar si el programa de alimentación debe estar activo según los parámetros introducidos por el usuario, para ello hay que tener en cuenta la hora del sistema.

```
public void comprobarsilos(){
    java.util.Calendar c = java.util.Calendar.getInstance();
    hora = c.get(java.util.Calendar.HOUR_OF_DAY);
```

```
    minuto = c.get(java.util.Calendar.MINUTE);
    segundo = c.get(java.util.Calendar.SECOND);

    if ((hora>=h1)&&(hora<h)){
        siloactivado();
    }else{
        seekBarretirada.setVisibility(INVISIBLE);
        seekBarengorde.setVisibility(INVISIBLE);
        seekBarmadres.setVisibility(INVISIBLE);
        Global.setSilos(false);
    }if ((hora>=h1)&&(hora==h)&&(minuto<m)){
        siloactivado();
    }
    //si el modo está en manual
    if(!comprobarmodo()){
        siloactivado();
    }
}
```

La función *siloactivado()* tiene como objetivo comprobar si alguno de los *Switch* está encendido, es decir, cuál de los tres piensos quiere el ganadero suministrar. Lo que ocurre es que se deshabilitan el resto de los silos y se almacena cual es el pienso elegido.

Uno de los requerimientos que hay que cumplir es que solo se puede suministrar un tipo de pienso a la vez, por lo que para ello hemos programado el siguiente código del que solo se muestra un ejemplo:

```
public void onclick ( View view ){
    if (view.getId()==R.id.switch_madres){
        if((switchmadres.isChecked())&&(!comprobarmodo())){
            siloactivado();
        }if (!switchmadres.isChecked()){
            seekBarmadres.setVisibility(INVISIBLE);
        }if((switchmadres.isChecked())&&(comprobarmodo())){
            switchengorde.setChecked(false);
            switchretirada.setChecked(false);
            seekBarengorde.setVisibility(INVISIBLE);
            seekBarretirada.setVisibility(INVISIBLE);
            comprobarsilos();
        }
    }
...
```

Cada vez que alguno de los *Switch* sea seleccionado, el programa se asegurará de deshabilitar los otros dos *Switch*, así como la *SeekBar* correspondiente.
El primer *if* hace referencia al momento en el que el *Switch* está activo y el modo es manual.
El segundo *if* hace referencia al momento en el que se apaga el *Switch*.
En el último nos encontramos en el caso en que el *Switch* está activo y el modo es el automático.

Para introducir las horas y asegurarnos que se muestran en el formato correcto, se han utilizado de nuevo los *TimePicker* mencionados en la ventana de iluminación en el apartado 6.3.6. de la página 69.

En cuanto al dibujo de los silos, se ha utilizado de nuevo la librería *Canvas*. Esta vez se ha programado el código para simular que el pienso va disminuyendo según el usuario mueva hacia la derecha la *SeekBar* que tiene asociada cada uno de ellos. Cuando se ha disminuido al máximo el nivel del pienso debe aparecer un mensaje de alarma que indique que el pienso es insuficiente y cuál es el tipo de pienso que se está acabando. Para ello se ha implementado un *AlertDialog* cuyo código se ha explicado previamente en el apartado 6.3.7. de ventilación.

### 6.3.11. Ventana datos técnicos

Una vez se accede a la ventana de datos técnicos lo primero que se puede observar es una lista con las diferentes bandas creadas. Esta lista es un elemento llamado *RecyclerView*. Para poder utilizar este elemento, así como el *CardView*, se ha tenido que añadir las dependencias al sistema de construcción Gradle. Para la implementación de este elemento se ha creado la clase *RecyclerTouchListener* y la clase *BandasAdaptador* mostrados a continuación:

Como se define un *Adapter*:

- Contiene una clase interna *ViewHolder*, que permite obtener referencias de los componentes visuales (views) de cada elemento de la lista,
- Presenta un constructor y/o métodos para gestionar el Data Set (añadir, editar o eliminar elementos),
- Contiene un método *onCreateViewHolder* que infla el *layout* (archivo xml) que representa a nuestros elementos, y devuelve una instancia de la clase *ViewHolder* que antes definimos;
- Contiene un método *onBindViewHolder* que enlaza nuestra data con cada *ViewHolder*, y
- Contiene un método *getItemCount* que devuelve un entero indicando la cantidad de elementos a mostrar en el *RecyclerView*.

```
public class BandasAdaptador extends
RecyclerView.Adapter<BandasAdaptador.ViewHolderbanda> implements
View.OnClickListener{
    private View.OnClickListener listener;
    private List<claseBandas> bandaLista;

    public BandasAdaptador(List<claseBandas> bandaLista)
    {this.bandaLista = bandaLista;}

    public void setListaDeBandas(List<claseBandas> banda)
    {this.bandaLista = banda;}

  @NonNull
  @Override
 public ViewHolderbanda onCreateViewHolder(@NonNull ViewGroup viewGroup, int i)
{
   //creamos una nueva lista
        View filabanda =
LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.item_banda,
viewGroup, false);
        filabanda.setOnClickListener(this);
        return new ViewHolderbanda(filabanda);
    }
//Este método asigna valores para cada elemento de la lista
    @Override
    public void onBindViewHolder(@NonNull ViewHolderbanda viewHolderbanda, int
position) {

viewHolderbanda.id.setText(String.valueOf(bandaLista.get(position).getCodigo())
);

viewHolderbanda.fechaBanda.setText(String.valueOf(bandaLista.get(position).getF
echainicio()));

viewHolderbanda.numeroBanda.setText(String.valueOf(bandaLista.get(position).get
NumeroBanda()));
viewHolderbanda.estadoBanda.setText(String.valueOf(bandaLista.get(position).get
EstadoBanda()));
```

```
    }
//Método que define la cantidad de elementos del RecyclerView
@Override
    public int getItemCount() {
        return bandaLista.size();
    }

    @Override
    public void onClick(View view) {
        if(listener != null){
            listener.onClick(view);}
    }
// Clase interna que permite obtener referencias de los componentes visuales de
cada elemento de la lista.
    static class ViewHolderbanda extends RecyclerView.ViewHolder{

        TextView id,fechaBanda,numeroBanda,estadoBanda;

        ViewHolderbanda (@NonNull View itemView){
            super(itemView);
            id =  itemView.findViewById(R.id.miembro_id);
            fechaBanda =  itemView.findViewById(R.id.fechaBanda);
            numeroBanda =  itemView.findViewById(R.id.numeroBanda);
            estadoBanda = itemView.findViewById(R.id.estadoBanda);

        }
    }
}
```

Los métodos más importantes de la clase *RecyclerTouchListener* son los siguientes:

```
public RecyclerTouchListener(Context context, final RecyclerView recyclerView, final
ClickListener clickListener) {
    this.clickListener = clickListener;
    this.gestureDetector = new GestureDetector(context, new
GestureDetector.SimpleOnGestureListener() {
        @Override
        public boolean onSingleTapUp(MotionEvent e) {
            return true;
        }

        @Override
        public void onLongPress(MotionEvent e) {
            View child = recyclerView.findChildViewUnder(e.getX(), e.getY());
            if (child != null && clickListener != null) {
                clickListener.onLongClick(child,recyclerView.getChildPosition(child));
            }
        }
    });
}

@Override
public boolean onInterceptTouchEvent(@NonNull RecyclerView recyclerView, @NonNull
MotionEvent motionEvent) {
    View child = recyclerView.findChildViewUnder(motionEvent.getX(), motionEvent.getY());
    if (child != null && clickListener != null &&
gestureDetector.onTouchEvent(motionEvent)) {
        clickListener.onClick(child, recyclerView.getChildAdapterPosition(child));
    }
    return false;
}
```

...

En el archivo .java de la ventana datos técnicos de las bandas es donde se obtiene la instancia del *RecyclerView*, se crea el *Layoutmanager*, se generará una lista con el *item_banda* y finalmente se crea el adaptador que coordinará los elementos.

```
listabandas = new ArrayList<>();
bandasAdaptador = new BandasAdaptador(listabandas);
recyclerViewBandas=findViewById(R.id.RecyclerViewBandas);
recyclerViewBandas.setLayoutManager(new LinearLayoutManager(this));
recyclerViewBandas.setItemAnimator(new DefaultItemAnimator());
recyclerViewBandas.setAdapter(bandasAdaptador);
```

Una vez configurado el *RecyclerView* le cargamos los datos de la base de datos llamando a la función *refrescarListadebandas()* cuyo código se muestra a continuación. La función utilizada para obtener los datos se mencionó anteriormente en el apartado 6.3.1.

```
public void refrescarListadebandas(){
    if (bandasAdaptador == null) return;
    listabandas = dbconeccion.obtenerbandas();
    Collections.reverse(listabandas);
    bandasAdaptador.setListaDeBandas(listabandas);
    bandasAdaptador.notifyDataSetChanged();
}
```

Se programa el *listener* para los clics en el *RecyclerView* de la siguiente manera:

```
recyclerViewBandas.addOnItemTouchListener(new
RecyclerTouchListener(getApplicationContext(), recyclerViewBandas, new
RecyclerTouchListener.ClickListener() {
    @Override // Un toque sencillo
    public void onClick(View view, int position) {
        // Pasar a la actividad ventanaeditarbanda.java
        bandaSeleccionada = listabandas.get(position);
        Intent intent = new
Intent(ventanadatostecnicosbandas.this,ventanaeditarbanda.class);
        intent.putExtra("codigo",bandaSeleccionada.getCodigo());
        intent.putExtra("fechabanda", bandaSeleccionada.getFechainicio());
        intent.putExtra("numero", bandaSeleccionada.getNumeroBanda());
        intent.putExtra("estadobanda", bandaSeleccionada.getEstadoBanda());
        startActivityForResult(intent,1);
    }


   @Override // Un toque largo
   public void onLongClick(View view, int position) {
       final claseBandas bandaParaEliminar = listabandas.get(position);
      AlertDialog dialog = new AlertDialog
              .Builder(ventanadatostecnicosbandas.this)
              .setPositiveButton("Sí, eliminar", new DialogInterface.OnClickListener() {
                  @Override
                  public void onClick(DialogInterface dialog, int which) {
                      dbconeccion.eliminarbanda(bandaParaEliminar);
                      refrescarListadebandas();
                  }
              })
              .setNegativeButton("Cancelar", new DialogInterface.OnClickListener() {
                  @Override
```

```
                    public void onClick(DialogInterface dialog, int which) {
                        dialog.dismiss();
                    }
                })
                .setTitle("Confirmar")
                .setMessage("¿Eliminar a la banda " + bandaParaEliminar.getCodigo() + "?")
                .create();
        dialog.show();

    }
}));
```

Del mismo modo cuando se regresa de nuevo a la actividad *ventanadatostecnicosbanda* se devuelve el dato estado de la banda para poder actualizarla en la base de datos en caso de que su valor se haya modificado.
El código desarrollado para recibir un resultado de una actividad es el siguiente:

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    super.onActivityResult(requestCode, resultCode, data);
    // check if the request code is same as what is passed  here it is 2
    if(requestCode==1)
    {if(resultCode==RESULT_OK){
        int
estado=data.getIntExtra("estado",bandaSeleccionada.getEstadoBanda());
        claseBandas bandaeditada = new
claseBandas(bandaSeleccionada.getCodigo(),bandaSeleccionada.getFechainicio(),ba
ndaSeleccionada.getNumeroBanda(),estado);
        dbconeccion.editarbanda(bandaeditada);
    }}
}
```

El segundo método se ejecuta cuando el usuario mantiene pulsada cualquiera de las bandas, el programa dará al usuario la posibilidad de eliminar la banda seleccionada o cancelar la acción en caso de que se haya realizado un toque largo por equivocación. En el caso de eliminar la banda refrescaremos la lista para que se muestren los datos correctos.

**Ventana Nueva Banda**
Para poder crear una banda nueva se debe clicar en el botón de agregar bandas de la parte inferior de la pantalla, esta función crea una intención que dirige al usuario a una nueva ventana llamada *ventananuevabanda* en la que aparecen los datos ya rellenados, aunque el usuario puede modificarlos. Para modificar el día se ha hecho uso de un *DatePickerDialog* cuyo funcionamiento muy parecido al *TimePicker* ya explicado con anterioridad.

A la hora de crear la banda lo que se hace es crear un nuevo objeto de tipo *claseBandas* introduciendo los datos que hay en los respectivos *EditText.* A continuación, se invoca al método *nuevaBanda()* que se encuentra en la clase *SQLControlador*, pasándole el objeto creado como se puede ver en el siguiente trozo de código:

```
        claseBandas nuevaBanda = new
claseBandas(1,fecha,Integer.parseInt(dia),Integer.parseInt(estado));

        long id = dbconeccion.nuevaBanda(nuevaBanda);
```

**Ventana Editar Banda**
Cuando accedemos a la ventana de edición de las bandas lo primero que se hace es leer los extras que se han recibido.

```
id = getIntent().getExtras().getInt("codigo");
fecha = getIntent().getExtras().getString("fechabanda");
numero = getIntent().getExtras().getInt("numero");
estado = getIntent().getExtras().getInt("estadobanda");
returnIntent = getIntent();
```

A continuación, se invoca la función *consultarBanda(id)* la cual tiene como dato de entrada el *id* de la banda seleccionada por el usuario. Lo que se hace dentro de esta función es acceder a la *TABLA_DETALLESBANDAS* para leer el valor de los datos almacenados y mostrarlos en pantalla.

Si se selecciona el botón "guardar cambios" se ejecuta un método encargado de acceder a la base de datos para guardar todos los campos que se muestran en pantalla.

Por último, el botón "Resumen Banda" redirigirá al usuario a una nueva ventana en la que se muestran los datos técnicos más importantes y significativos. De esta manera el usuario e incluso el veterinario pueden utilizar los resultados para comparar y mejorar el rendimiento de la explotación. El código desarrollado en esta actividad consiste simplemente en consultar la base de datos y cargar los datos en pantalla calculando correctamente cada uno de los parámetros.

### 6.3.12. Ventana estadísticas
En la ventana estadísticas el usuario tiene la posibilidad de graficar cualquiera de los datos técnicos de cualquiera de las bandas y además puede visualizar en una gráfica lineal el valor que toman cualquiera de los sensores instalados en la explotación a lo largo del tiempo.
Al comenzar la actividad se establece la conexión con la base de datos y se obtiene una lista con la cantidad de bandas que existen. Después se recorre dicha lista y se rellena un *ArrayList* con los valores que deben aparecer en el *Multispinner*. Seguidamente se rellena el *Spinner* con las bandas y los datos que se van a poder graficar. Esta parte del código no se muestra por ser repetitiva.

```
dbconeccion = new SQLControlador(this);
obtenerBandas();
valores = new ArrayList<>(listabandas.size());
for(int i = 1 ; i < listabandas.size() ; i++) {
    valores.add("Banda "+i);
}
multiSpinnerBandas.setItems(valores,
        "Selecciona Bandas", this);
...

multiSpinnerDatos.setItems(datos,
        "Selecciona Datos", this);
```

Cuando se rellenan los datos de los Spinners y se pulsa el botón "Graficar", se dirige al usuario a una nueva actividad a la que se le envían los datos seleccionados, es decir, las bandas seleccionadas, los datos seleccionados, el tipo de grafico escogido y una variable llamada sensor, que sirve para identificar cuál de los sensores queremos graficar. En el caso de graficar los datos técnicos de las bandas, el valor de esta variable será igual a cero.

El tipo de gráfico que se quiere graficar puede ser de líneas o de barras, en esta parte del código se observa cómo se ha determinado la manera de almacenar el tipo elegido. De modo que la variable booleana tomará un valor verdadero cuando el grafico sea de líneas y negativo cuando se trate de un gráfico de barras.

```
if(rbLineas.isChecked()){
    tipodegrafico=true;
}if(rbBarras.isChecked()){
    tipodegrafico=false;
```

En este proyecto, al ser un prototipo, a la hora de guardar los datos procedentes de las sondas, se ha simulado utilizado un archivo de texto para introducir datos aleatorios, los cuales han sido posteriormente introducidos en una base de datos para finalmente ser leídos y mostrados en pantalla en la actividad graficar.

La siguiente función es el modo de acceder a un archivo almacenado en el directorio .res en la carpeta .raw y leer todos los valores e introducirlos en la base de datos en su tabla correspondiente. Para ello se ha creado esta función a la que hay que introducirle el nombre de la variable elegida, el parámetro y la tabla a consultar. Esta función solo se ejecutará una vez para cada uno de los archivos.

```
private void Inicializar(String variable, String var,String tabla,int sensor){
    Intent intent = new Intent(this, graficar.class);
    if(cantidad(variable)==0){
        String[] texto = leerArchivo(variable);
        dbhelper = new MyOpenHelper(this,NOMBRE_BD,null,1);
        db = dbhelper.getWritableDatabase();
        db.beginTransaction();
        for(int i=0;i<texto.length;i++){
            String [] linea = texto[i].split(";");
            ContentValues contentValues = new ContentValues();
            contentValues.put("id",linea[0]);
            contentValues.put(var,linea[1]);
            db.insert(tabla,null,contentValues);
        }
        Toast.makeText(this,"Datos
insertados"+texto.length,Toast.LENGTH_LONG).show();
        db.setTransactionSuccessful();
        db.endTransaction();

        intent.putExtra("bandasseleccionadas", (String[]) null);
        intent.putExtra("tipodegrafico", true);
        intent.putExtra("datosseleccionados", (String[]) null);
        intent.putExtra("sensores",sensor);
        startActivity(intent);
    }
...
```

*LeerArchivo* es la función que accede al archivo .txt en la que hay un *Switch* con los diferentes nombres de los archivos.
A continuación, se muestra un ejemplo:

```
case "temperaturainterior":
     inputStream = getResources().openRawResource(R.raw.temperaturainterior);
     break;
```

En la figura 33 podemos observar cinco botones, uno para cada sonda. Estos botones son los encargados de graficar los correspondientes datos de cada sensor. Según cuál de ellos se pulse, se invocará la función anterior *Inicializar()* introduciendo los valores correspondientes. A continuación, se muestra el ejemplo de la temperatura interior:

```
public void graficartemperaturainterior (View view){
    Inicializar("temperaturainterior","tempint","TABLA_TINT",1);
}
```

Cualquier gráfica, ya sea datos de los sensores o datos técnicos de las bandas serán mostrados en la actividad Graficar que se explicará a continuación:

**Activity Graficar**

La primera acción que se realiza al crear la actividad graficar es recoger los datos provenientes de la actividad que ha realizado la intención (bandas y datos seleccionados, tipo de gráfico y sensor). Cuando la variable sensor sea distinta a cero, es decir, cuando se pretendan graficar datos de las sondas, la función que se invocará será *graficarsensores()* que lo que hace es crear un cursor y un ArrayList de tipo <Entry> . En el cursor se guardan los valores adecuados, accediendo a la tabla de la base de datos correspondiente. Después se recorrerá dicho cursor para añadir cada entrada al ArrayList y seguidamente graficar los valores. A continuación, se muestra una parte del código:

```
ArrayList<Entry> entries = new ArrayList<>();
switch(sensores)
{
    case 1:
        dato = BaseDeDatabase.rawQuery("select tempint from temperaturainterior",
null);
        break;

...

if (dato.moveToFirst()) {
    //Recorremos el cursor hasta que no haya más registros
    do {
        float datosensor = Float.parseFloat(dato.getString(0));
        entries.add(new Entry(i,datosensor));
        i++;
    } while(dato.moveToNext());
}dato.close();
BaseDeDatabase.close();
LineDataSet dataset = new LineDataSet(entries, "Datos de sensorización");
LineData data = new LineData(dataset);
graficaLineas.setData(data);
}
```

Del mismo modo se grafican los datos técnicos de las bandas. Según si se ha seleccionado un tipo de gráfico u otro, las entradas serán diferentes. Es decir, puede ser una *ListChart* o una *BarChart* para las cuales se utilizan respectivamente *ArrayList* de *List<BarEntry>* o *List<Entry>.*

# 7. Pruebas y ajustes finales o de servicio.

A lo largo del desarrollo de la aplicación se han realizado varias pruebas mediante las cuales se ha analizado el proceso de ejecución y se ha comprobado que se cumplen los requisitos. Se han llevado a cabo depuraciones de código a través de un emulador, lo que ha permitido detectar y solucionar diversos errores de código.
Además, se han realizado diferentes pruebas concretas para cada una de las diferentes ventanas, que permiten garantizar que el programa funciona correctamente. Estas se describen a continuación:

| Prueba 1 | General |
|---|---|
| Objetivo | Acceso a la aplicación |
| Descripción | Se introducirán las credenciales (nombre y contraseña) para acceder a la aplicación, comprobando además que, si se introducen las credenciales incorrectas, aparece un mensaje de error. |
| Resultado | Superada |

Tabla 1: Prueba 1 (General)

| Prueba 2 | General |
|---|---|
| Objetivo | Apertura y navegación correcta en la aplicación |
| Descripción | Se abrirá la aplicación y se navegará por todas sus partes comprobando que la aplicación y los botones de la fecha hacia atrás funcionan correctamente. Además, se comprobará el acceso a ambas naves para las diferentes pantallas de la aplicación. |
| Resultado | Superada |

Tabla 2: Prueba 2 (General)

| Prueba 3 | General |
|---|---|
| Objetivo | Verificar que se guardan los parámetros editables |
| Descripción | Verificar para cualquier ventana que, si se modifica cualquier parámetro editable, al cerrar la ventana y volverla a abrir, el dato se habrá guardado en preferencias y se podrá visualizar en pantalla. |
| Resultado | Superada |

Tabla 3: Prueba 3 (General)

| Prueba 4 | General |
|---|---|
| Objetivo | Comprobar el acceso las diferentes naves. |
| Descripción | Dentro de cada ventana se localiza en la parte inferior derecha un botón que permite al usuario acceder al programa de la otra nave. Se comprobará que desde cualquier ventana podemos acceder a la nave 1 y a la nave 2. |
| Resultado | Superada |

Tabla 4: Prueba 4 (General)

| Prueba 5 | Estado General |
|---|---|
| Objetivo | Acceso a las diferentes pantallas desde estado general. |
| Descripción | Se comprueba el acceso a las diferentes pantallas de la aplicación pulsando sus dibujos correspondientes desde la pantalla estado general. |
| Resultado | Superada |

Tabla 5: Prueba 5 (Estado general)

| Prueba 6 | Iluminación |
|---|---|
| Objetivo | Comprobar que funciona correctamente el modo manual y el modo automático. |
| Descripción | Cuando el modo manual está activado el usuario puede apagar y encender la iluminación de la nave con pulsar el dibujo de la bombilla. Sin embargo, en el modo automático el programa es el encargado de calcular según los datos introducidos si la iluminación estará encendida o apagada. |
| Resultado | Superada |

Tabla 6: Prueba 6 (Iluminación)

| Prueba 7 | Iluminación |
|---|---|
| Objetivo | Comprobar el funcionamiento de la gráfica. |
| Descripción | Se comprobará que, al modificar el día del ciclo, o alguno de los parámetros que aparecen en la flecha inferior (inicio y fin de "flushing", fin rampa, inicio y fin sin luz y fin de ciclo), la gráfica se modificará coloreando de verde la parte de la gráfica en la que se encuentra la nave según los parámetros introducidos. |
| Resultado | Superada |

Tabla 7: Prueba 7 (Iluminación)

| Prueba 8 | Ventilación |
|---|---|
| Objetivo | Comprobar el programa de carga de kilos de peso vivo. |
| Descripción | Se comprobará que el número de gazapos y madres que el programa calcula que hay a partir de los datos que introduce el usuario son correctos. El número de gazapos será igual al número de camadas multiplicado por el número de gazapos por camada. Además, si el usuario introduce en los datos que no hay camadas e introduce un numero distinto de cero para el dato de numero de gazapos por camadas, aparecerá un mensaje en pantalla que advierta al usuario que eso no es posible e introducirá el cero automáticamente. |
| Resultado | Superada |

Tabla 8: Prueba 8 (Ventilación)

| Prueba 9 | Ventilación |
|---|---|
| Objetivo | Comprobar el aviso de encendido de “cooling” y grupo de apoyo. |
| Descripción | Cuando la temperatura interior exceda 1,5 ºC la temperatura máxima, se encenderá el “cooling” y aparecerá en pantalla un *AlertDialog* que nos dará la opción de acceder a la ventana de refrigeración para ver el estado del “cooling”. En cuanto al grupo de apoyo, éste se encenderá cuando la temperatura alcance la temperatura mínima establecida en la pantalla grupos. Y de nuevo aparecerá un *AlertDialog* con el mismo objetivo. |
| Resultado | Superada |

Tabla 9: Prueba 9 (Ventilación)

| Prueba 10 | Ventilación |
|---|---|
| Objetivo | Comprobar el acceso a grupos y el correcto funcionamiento del programa. |
| Descripción | Comprobar que cuando la temperatura interior es inferior a la temperatura mínima, el porcentaje actual será cero y la gráfica no aparecerá coloreada.<br>Cuando la temperatura alcance la temperatura mínima, el grupo de apoyo debe haberse encendido, calculando el porcentaje y coloreando la gráfica en la sección correspondiente.<br>Si la temperatura excede la temperatura de alarma, debe aparecer un mensaje que advierta al usuario de que la temperatura es demasiado elevada. |
| Resultado | Superada |

Tabla 10: Prueba 10 (Ventilación)

| Prueba 11 | Calefacción |
|---|---|
| Objetivo | Correcto funcionamiento del cañón de propano. |
| Descripción | Se modificará la temperatura interior para observar si el cañón de propano funciona correctamente. También se modificará el valor de los segundos de retardos para asegurarnos de que efectivamente el cañón tarda en encenderse y apagarse dicho valor en segundos. |
| Resultado | Superada |

Tabla 11: Prueba 11 (Calefacción)

| Prueba 12 | Refrigeración |
|---|---|
| Objetivo | Correcto funcionamiento del “cooling” y programa de refrigeración. |
| Descripción | Se modificará la temperatura interior para comprobar el correcto funcionamiento del programa de refrigeración. Según la temperatura introducida deben variar el porcentaje actual de la bomba y la gráfica que será coloreada según en la zona en la que se encuentre.<br>Además, se modificará el valor de los porcentajes de la bomba, para comprobar que cuando se introduce el 100 % del porcentaje, la gráfica cambia de forma que se divide en tres zonas. |
| Resultado | Superada |

Tabla 12: Prueba 12 (Refrigeración)

| Prueba 13 | Alimentación |
|---|---|
| Objetivo | Correcto funcionamiento de las alarmas. |
| Descripción | Se comprobará que si la cantidad de pienso en cada uno de los silos es insuficiente la alarma avisará al usuario, para ello se moverá hacia la derecha cada uno de los *Scrollbars* que se encuentran debajo de los silos. Debe aparecer un *AlertDialog* que adviertan que el pienso es insuficiente y cuál de los tres piensos se está acabando. |
| Resultado | Superada |

Tabla 13: Prueba 13 (Alimentación)

| Prueba 14 | Alimentación |
|---|---|
| Objetivo | Comprobar que funciona correctamente el modo manual y el modo automático. |
| Descripción | Cuando el modo manual está activado el usuario puede activar y desactivar los silos de la nave con pulsar el *Switch* de cada uno de ellos. Sin embargo, en el modo automático el programa es el encargado de calcular si el silo debe estar activado o desactivado según los datos introducidos, únicamente el usuario elige que tipo de pienso quiere administrar a los animales.<br>Además, se deberá comprobar que el programa permite tener únicamente un silo activado. |
| Resultado | Superada |

Tabla 14: Prueba 14 (Alimentación)

| Prueba 15 | Datos técnicos de las bandas |
|---|---|
| Objetivo | Creación de nueva banda |
| Descripción | Comprobar que al pulsar el botón nueva banda y rellenar los campos correspondientes, aparece en la ventana principal de datos técnicos la nueva banda con sus correspondientes datos. |
| Resultado | Superada |

Tabla 15: Prueba 15 (Datos técnicos de las bandas)

| Prueba 16 | Datos técnicos de las bandas |
|---|---|
| Objetivo | Edición de una banda |
| Descripción | Si se pulsa en cualquiera de las bandas aparece un formulario con los datos técnicos de las bandas para rellenar o editar.<br>Modificar varios datos, pulsar el botón guardar cambios y comprobar que efectivamente si volvemos acceder a la ventana de editar datos, los campos modificados han sido guardados en la base de datos y por tanto se muestran en pantalla. |
| Resultado | Superada |

Tabla 16: Prueba 16 (Datos técnicos de las bandas)

| Prueba 17 | Datos técnicos de las bandas |
|---|---|
| Objetivo | Eliminar una banda |
| Descripción | Comprobar que, si se mantiene pulsado (con un clic largo) cualquiera de las bandas, el programa nos da la opción de eliminarla de la base de datos. |
| Resultado | Superada |

Tabla 17: Prueba 16 (Datos técnicos de las bandas)

| Prueba 18 | Estadísticas |
|---|---|
| Objetivo | Comprobar que se grafican de datos correctamente. |
| Descripción | Para ello se seleccionará varios datos aleatoriamente y se comprobará que el programa grafica correctamente tomando los valores correspondientes. Además, se comprobará que según el tipo de gráfico que se seleccione aparecerá un gráfico de barras o uno de líneas.<br>También se comprobará que se puede tener acceso a los datos arrojados por las diferentes sondas pulsado sus respectivos botones y observando cómo se crea una gráfica lineal con los datos correspondientes. |
| Resultado | Superada |

Tabla 18: Prueba 18 (Estadísticas)

# 8. Presupuesto

En este apartado se procede a detallar el presupuesto del proyecto. En el presupuesto se ha tenido en cuenta el gasto en personal, así como los gastos de recursos software y hardware necesarios para desarrollar la aplicación.

A continuación, se hace una estimación de las horas que se han invertido en cada fase del proyecto, así como del precio por hora correspondiente a cada uno de los profesionales que toman parte de este proyecto. En cuanto a los recursos hardware y software introduciremos en el presupuesto el coste proporcional al uso que se le ha dado en el proyecto.

Las horas empleadas en las diferentes fases de proyecto han sido aproximadamente las siguientes:

- Análisis y diseño: 150 horas.
- Implementación: 430 horas.
- Documentación: 90 horas.

Será necesaria la colaboración de un analista que desempeñe las tareas de análisis y diseño del proyecto. Este tipo de profesionales cobran 15 € la hora. También ha sido imprescindible un ingeniero informático a modo de programador que se encargue de realizar todo el desarrollo de la aplicación. El coste por hora es de 14 €. Por último, el documentalista es el encargado de realizar todo lo relacionado con la redacción de la memoria y cuyo coste por hora trabajada es de 10 € la hora.

A continuación, se calcula el coste en recursos humanos distinguiendo entre las diferentes tareas a realizar a lo largo del proyecto.

| Descripción | Horas | Coste (€/h) | Coste total (€) |
|---|---|---|---|
| Analista | 150 | 15 | 2250 |
| Programador | 430 | 14 | 6020 |
| Documentalista | 90 | 10 | 900 |
| TOTAL | 720 | | 9170 |

Tabla 19: Presupuesto. Coste de personal

Presupuesto de Mano de Obra = horas Analista · precio Analista + horas Documentalista · precio Documentalista + horas Programador · precio Programador = 9170 €

| Hardware | Uso (%) | Coste total (€) | Coste proporcional (€) |
|---|---|---|---|
| Ordenador portátil | 20 | 650 | 130 |
| Wifi | 15 | 50 | 7,5 |
| Dispositivo Android | 10 | 300 | 30 |
| TOTAL | | | 167,5 |

Tabla 20: Presupuesto. Coste de hardware

Presupuesto Hardware = 167,5 €

| Software | Uso (%) | Coste total (€) | Coste proporcional (€) |
|---|---|---|---|
| Office 365 | 100 | 49,5 | 49,5 |
| TOTAL | | | 49,5 |

Tabla 21: Presupuesto. Coste de software

Presupuesto Software = 74,5 €

| | Coste total (€) |
|---|---|
| Personal | 9170 |
| Hardware | 167,5 |
| Software | 49,5 |
| TOTAL | **9387** |

Tabla 22: Presupuesto. Coste total

Finalmente, si sumamos todos los costes del proyecto, el precio final de los costes alcanza los 9387 euros.

# 9. Conclusiones

## 9.1. Valoración personal

El trabajo final de grado representa el fin de los estudios y el principio de una etapa en la que se comenzará a aplicar todos los conocimientos aprendidos y capacidades adquiridas.
Desde hace unos años tenía curiosidad por implementar una aplicación para teléfonos móviles, ya que es una tecnología en auge y empleada por una gran cantidad de usuarios, sin embargo, no me animé a dar el paso, (quizás por falta de tiempo o miedo a enfrentarme a algo nuevo) hasta que vi en este trabajo final la oportunidad de dedicar suficiente tiempo para aprender y desarrollar una aplicación decente.

El hecho de emprender este proyecto, basado en un lenguaje de programación y una tecnología de las cuales no tenía apenas conocimiento, ha supuesto un reto para mí.
Cuando comencé con el proyecto, me pareció una buena idea adentrarme un poco más en el lenguaje de programación Java y Android, por ese motivo, tomé algunos libros prestados como *El gran libro de Android* (Gironés, 2018). También aproveché la oportunidad de que la universidad tenía un convenio con edX (plataforma que trabaja con las universidades y organizaciones líderes a nivel mundial para ofrecer cursos en línea de alta calidad a estudiantes de todo el mundo) y me animé a realizar uno de los cursos ofertados: Introducción a Android.
He de decir que comenzar aprendiendo de esta manera me facilitó mucho las cosas, ya que había muchos conceptos que no conocía y en el curso se explicaban de forma muy sencilla y didáctica.

Por lo tanto, he aprendido muchas cosas sobre Java y Android. Además, desarrollar una aplicación desde el principio ha hecho que tenga una nueva visión de la informática, ya que he aprendido la importancia de una buena planificación, diseño e identificación de los requisitos.

Realmente con la realización de este proyecto he podido completar mi formación haciendo frente a un problema real, donde iban surgiendo diferentes inconvenientes. De este modo, he mejorado mi capacidad para investigar y aportar soluciones para la resolución de dichos problemas, aportando mis conocimientos tecnológicos.

En este proyecto he disfrutado mucho trabajando y aprendiendo nuevos conocimientos, además también he desarrollado mi desenvoltura a la hora de enfrentar un problema. Ha sido duro pero satisfactorio. En definitiva, ha sido necesario emplear muchas horas, pero con actitud y abriendo caminos poco a poco, se iban viendo los resultados. Tras muchos meses de trabajo, he terminado mi proyecto, muy contenta por haberlo acabado, pero sobre todo por haber aprendido tanto a lo largo de este.

Por lo tanto, he cumplido satisfactoriamente mi objetivo personal con este proyecto, además se ha conseguido diseñar una aplicación para móviles fácil y elegante gracias a las guías de Material Design de Android. Esto junto a los tonos elegidos, el diseño, la inserción de gráficas, imágenes y títulos representativos, se ha creado una interfaz intuitiva y agradable para el usuario que le permitirá moverse por la aplicación con la mayor comodidad posible, ya que la información se muestra de forma muy limpia y ordenada a la vista.
Si analizamos su funcionalidad, comprobamos que se cumplen todos los objetivos y requisitos propuestos llevando a cabo el control tanto de los parámetros ambientales como de los datos técnicos de las bandas.

La introducción de esta nueva tecnología en el ámbito de las explotaciones cunícolas provoca una notoria mejoría, ya que con los datos arrojados por las diferentes sondas que miden los parámetros ambientales se consigue un mayor cuidado de los animales, provocando con ello un aumento de la producción cárnica. Además, se simplifica el control de stock, ya que con la

aplicación es posible que el granjero disponga de los piensos y agua necesarios para poder conservarlos con la mayor calidad posible.
Otro aspecto importante a tener en cuenta es la posibilidad de observar los datos (ambientales y técnicos) mediante gráficos, y compararlos para un mayor rendimiento y productividad de la nave.

## 9.2. Futuras ampliaciones o mejoras

La principal futura ampliación será transformar este proyecto el cual es un prototipo, en una aplicación real. Es decir, dicha mejora conllevaría la conexión de los distintos sensores.
Como ya se ha mostrado en apartados anteriores los sensores instalados en la explotación serán: temperatura, humedad, oxígeno, Co2, ultrasonidos para estado del agua (litros) y del pienso (kilos).

La arquitectura utilizada será la de cliente-servidor, se optará por un gestor de base datos como puede ser MySQL. Se utiliza Internet como red de conexión debido a que requiere que aplicación esté disponible en cualquier lugar.

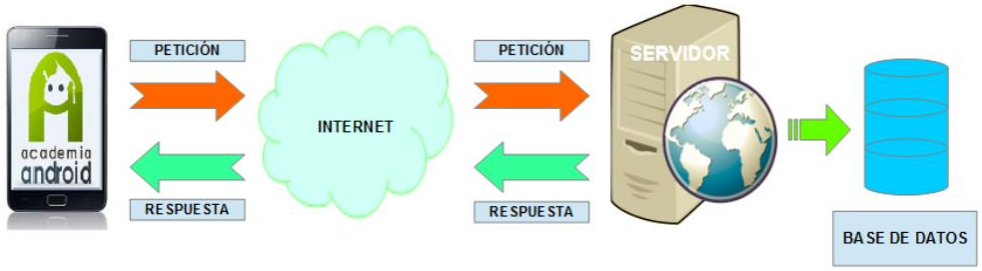


Figura 39: Arquitectura cliente-servidor (*Aplicaciones cliente-servidor y redes de telefonía móvil – Academia Android*, n.d.).

La aplicación Android desempeña las funciones de cliente y constituye la principal y única fuente de datos a través de la cual el usuario puede tener al alcance toda la información almacenada de forma remota. Se hará uso de un servidor remoto que regula el acceso a la base de datos donde se almacenan los datos que precisa la app y a otras herramientas necesarias para el correcto funcionamiento de esta. Así mismo, desde la propia aplicación Android se enviarán los comandos necesarios para accionar los actuadores.
Básicamente la aplicación se comunicará a una base de datos remota por medio de un web service que realiza las peticiones al servidor el cual valida la información y devuelve un request si el proceso fue satisfactorio o en caso contrario, si fue un proceso fallido, devuelve un mensaje de error.

Por otra parte, la mejor forma de probar la aplicación y observar qué mejoras se pueden incorporar, es sin duda que el usuario haga uso de esta durante un tiempo. Sin embargo, algunas de las propuestas para mejorar la funcionalidad de la aplicación podrían ser:

- Sería interesante añadir una ventana en la aplicación donde se pudiera introducir el peso exacto de los conejos. Es decir, el usuario puede pesar a los animales durante diferentes etapas para obtener un resultado más preciso. De este modo se puede mejorar la producción de kilos de carne y poder contar de un modo más rápido y sencillo la cantidad de kilos que finalmente son vendidos al matadero.
- También sería una buena idea añadir una parte de gestión de usuarios y contraseñas en un servidor de Login. En este proyecto como es un prototipo no tiene sentido, sin embargo, para una futura mejora se podrían programar dos tipos de administración de la

aplicación: administrador y visitante. De este modo uno de los perfiles tendría acceso a visualizar y modificar cualquier parámetro mientras que el otro únicamente podrá acceder a la aplicación para visualizar lo que está ocurriendo y consultar datos. Habría además que programar un interfaz para el registro de los perfiles y la recuperación de contraseña.

- Otra mejora relacionada con la automatización de nave podría ser que se programara la ventana de alimentación para que según el día del ciclo en el que se encuentre la explotación, se seleccionara automáticamente un silo u otro. Para ello habría que introducir datos de la duración de cada etapa dentro del ciclo y permitir que sean modificables para el usuario.
- Con relación a los silos, se podría programar la aplicación para que cuando el nivel de pienso sea insuficiente, se rellene automáticamente un albarán y se envíe por correo electrónico a la empresa encargada de suministrarlo. De esta manera se facilita el trabajo al usuario, evitando cualquier posible incidente y ahorrando tiempo.
- En cuanto a los datos técnicos de las bandas, se podría aumentar la cantidad de datos que se almacenan, proporcionando de este modo más información al ganadero. Hay muchos parámetros que pueden ser de utilidad para llevar el control de la productividad de la explotación. Por ejemplo, se podría añadir una sección para que el usuario introduzca el motivo de baja de las conejas y gazapos. De este modo, se podrá observar a lo largo del tiempo que enfermedades son las que más afectan y qué espacio de tiempo.
- Instalación de más sensores, por ejemplo, un sensor para medir el propano del cañón de la calefacción.

# 10.Bibliografía


Ramírez, P. (2019, December 22). *¿Cuáles son los sistemas operativos más usados o utilizados en 2019? - ITSoftware*. https://itsoftware.com.co/content/sistemas-operativos-mas-usados/

Collado, C. (2019, October 7). *En Android se descargan más apps que en iOS y otros datos curiosos*. Retrieved August 18, 2020, from https://andro4all.com/2019/10/android-vs-ios-descargas-apps-juegos-estadisticas

*Máster en Desarrollo de Aplicaciones Android - Componentes de una aplicación*. (2017). http://www.androidcurso.com/index.php/recursos/31-unidad-1-vision-general-y-entorno-de-desarrollo/98-comparativa-con-otras-plataformas

GoogleDevelopers. (2019). *Panel de distribución | Desarrolladores de Android*. https://developer.android.com/about/dashboards?hl=es-419

Developers. (2020). *Arquitectura de la plataforma | Desarrolladores de Android*. Android Developers. https://developer.android.com/guide/platform?hl=es

*Pausar y Reanudar una Actividad - Gestionar el Ciclo de Vida de una Actividad*. (n.d.). Retrieved August 18, 2020, from https://desarrollador-android.com/desarrollo/formacion/empezar-formacion/gestionar-el-ciclo-de-vida-de-una-actividad/pausar-y-reanudar-una-actividad/

*Aplicaciones cliente-servidor y redes de telefonía móvil – Academia Android*. (n.d.). Retrieved September 2, 2020, from https://academiaandroid.com/aplicaciones-cliente-servidor-y-redes-de-telefonia-movil/

*Toni Roca : Artículos : Historia de la cunicultura industrial en España*. (n.d.). Retrieved September 2, 2020, from http://www.conejos-info.com/articulos/historia-de-la-cunicultura-industrial-en-espana

CEAC planeta Formación y Universidades, & Caros Yañez. (2017). *Diferencias entre apps nativas o híbridas | Ceac*. 19/07. https://www.ceac.es/blog/diferencias-entre-apps-nativas-o-hibridas

López Villegas, D. (2014). IDE: Entornos Integrados de Desarrollo para Android – Academia Android. In *Digital Learning SL*. https://academiaandroid.com/ide-entornos-integrados-de-desarrollo-para-android/

Android Studio. (2019). *Introducción a Android Studio | Desarrolladores de Android*. https://developer.android.com/studio/intro?hl=es-419

Universidad Politecnica de Valencia. edX. (n.d.). *Introducción a la plataforma para móviles Android | Introducción a la plataforma para móviles Android | Material del curso AIP201x | edX*. Retrieved August 18, 2020, from https://courses.edx.org/courses/course-v1:UPValenciaX+AIP201x+2T2019/courseware/Unidad3/Unidad3Subsection2Sequential/?activate_block_id=block-v1%3AUPValenciaX%2BAIP201x%2B2T2019%2Btype%40sequential%2Bblock%40Unidad3Su

bsection2Sequential

Universidad Politecnica de Valencia. (n.d.). *Máster en Desarrollo de Aplicaciones Android - Las versiones de Android y niveles de API*. Retrieved August 18, 2020, from http://www.androidcurso.com/index.php/146

*Componentes de una aplicación | Componentes de una aplicación | Material del curso AIP201x | edX*. (n.d.). Retrieved August 18, 2020, from https://courses.edx.org/courses/course-v1:UPValenciaX+AIP201x+2T2019/courseware/Unidad3/Unidad3Subsection12Sequential/?child=first

*Ficheros y directorios de un proyecto Android | Ficheros y directorios de un proyecto Android | Material del curso AIP201x | edX*. (n.d.). Retrieved August 18, 2020, from https://courses.edx.org/courses/course-v1:UPValenciaX+AIP201x+2T2019/courseware/Unidad3/Unidad3Subsection11Sequential/?child=last

Revelo, J. (2014). *Tutorial De Bases De Datos SQLite En Aplicaciones Android*. http://www.hermosaprogramacion.com/2014/10/android-sqlite-bases-de-datos/

*Ciclo de vida de una actividad | Ciclo de vida de una actividad | Material del curso AIP201x | edX*. (n.d.). Retrieved August 18, 2020, from https://courses.edx.org/courses/course-v1:UPValenciaX+AIP201x+2T2019/courseware/Unidad7/Unidad7Subsection2Sequential/?activate_block_id=block-v1%3AUPValenciaX%2BAIP201x%2B2T2019%2Btype%40sequential%2Bblock%40Unidad7Subsection2Sequential

Gironés, J. T. (2018). *El gran libro de Android* (6th ed.). Marcombo.

*Stack Overflow - Where Developers Learn, Share, & Build Careers*. (n.d.). Retrieved September 3, 2020, from https://stackoverflow.com/