UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA MECÁNICA

"DESARROLLO DE UNA APLICACIÓN DE AYUDA PARA PROPIETARIOS DE MASCOTAS CANINAS SOBRE EL SISTEMA OPERATIVO ANDROID"

TRABAJO FIN DE GRADO

Septiembre – 2021

AUTOR: Judith Gómez López

DIRECTOR: César Fernández Peris

ÍNDICE

Capítulo 1. INTRODUCCIÓN……4

1.1 Motivación y justificación del proyecto……..……..4

1.2 Sistema operativo Android: Mundo Android……...5

1.2.1 Historia: así nace Android……....5

1.2.2 Versiones del sistema……...6

1.3 Herramientas para desarrollar la aplicación para Android……13

1.3.1 MIT App Inventor 2……..13

1.3.2 Android Studio……..17

1.4 Aplicaciones similares……...……...20

1.5 Descripción de la aplicación……...……..24

Capítulo 2. MATERIAL Y MÉTODOS……..……...30

2.1 Adobe Photoshop...……...30

2.2 Android Studio……32

2.3 Android Manifest……..……33

2.4 Archivos xml……36

2.4.1 activity_main……38

2.4.2 activity_pets……39

2.4.3 activity_add……..……40

2.4.4 activity_details……47

2.4.5 activity_inf……47

2.4.6 activity_alarma……48

2.4.7 activity_illness……50

2.4.8 activity_contact……...54

2.5 Archivos java……..55

2.5.1 MainActivity……56

2.5.2 DBHelper……58

2.5.3 AddActivity……...61

2.5.4 BeanMascotas……73

2.5.5 PetsActivity……...78

2.5.6 DetailsActivity……..86

2.5.7 InfActivity……95
2.5.8 AlarmaActivity……97
2.5.9 Workmanagernoti……103
2.5.10 IllnessActivity……..108
2.5.11 ContactActivity ……...…117

Capítulo 3. RESULTADOS Y DISCUSIÓN……..121

Capítulo 4. CONCLUSIONES……122

Capítulo 5. BIBLIOGRAFÍA……...…123

Capítulo 6. TABLA DE ILUSTRACIONES……...124

CAPÍTULO 1. INTRODUCCIÓN

Este documento tiene como objetivo presentar y describir el proyecto realizado por la estudiante Judith Gómez López, con el fin de finalizar sus estudios en el Grado de Ingeniería Mecánica de la Universidad Miguel Hernández de Elche. En este proyecto participa César Fernández Peris, perteneciente al área de Telemática, como tutor del mismo.
Dicho proyecto consiste en el desarrollo de una aplicación móvil en el entorno del sistema operativo de Android. Para ello, se recurre a la herramienta *Android Studio.*
La aplicación en la cual se basa este proyecto, tendrá como finalidad el registro de diferentes mascotas caninas, sus respectivos cuidados y la posibilidad de compartir sus datos, además de mostrar información útil para sus dueños.

1.1 MOTIVACIÓN Y JUSTIFICACIÓN DEL PROYECTO

Hoy en día se llevan a cabo miles de trabajos con el fin de presentarlos para acabar varios años de estudio. Sin embargo, quería que el mío tuviera algo especial para mí, y es por ello que la idea de realizar este Trabajo de Fin de Grado viene inspirada por varios aspectos diferentes de mi vida.

Por un lado, he crecido en un entorno rodeada de diferentes tipos de animales, especialmente de perros. Mi familia siempre ha sido amante de ellos, llegando a acoger a muchos para poder encontrarles un hogar donde prime el cariño, el calor y el cuidado que éstos requieren. Para que esto dé buen resultado, uno de los aspectos más importantes es la organización: organización de sus datos, sus cuidados diarios, sus estudios veterinarios, etc.

Por otro lado, todos sabemos que en pleno siglo XXI la tecnología avanza muy rápidamente, y todos los avances tienen algo en común: la programación. Hay programación por todos lados y de muchos tipos, pero este trabajo se va a centrar en el desarrollo de una aplicación para el sistema operativo Android. Todo el mundo tiene en su *smartphone* aplicaciones móviles, es más, ya de

manera predeterminada te vienen las más básicas instaladas: notas, recordatorios, galería… todas ellas con su respectivo código interno, y su uso es tan cotidiano que ni siquiera nos paramos a pensarlo, pero, sin lugar a dudas, ayudan enormemente al ser humano. Sin embargo, es una enseñanza de la que Ingeniería Mecánica prácticamente carece, y es por ello que decido adentrarme un poco más en este campo, pues lo considero muy fructífero para mi futuro.

Por tanto, uniendo mi motivación a adentrarme en el mundo de la programación (específicamente al desarrollo aplicaciones) y mi conocimiento sobre la necesidad de una correcta organización cuando se tiene una o varias mascota/s canina/s, surgió la idea de desarrollar el proyecto que en este documento presento.

1.2 SISTEMA OPERATIVO ANDROID: MUNDO ANDROID

Como ya he presentado anteriormente, la aplicación ha sido desarrollada para el sistema operativo Android, pero antes haremos una breve introducción para que partamos en este proyecto bajo los mismos conocimientos.

Android es un sistema operativo desarrollado por Google y basado en el núcleo Linux y otros softwares de código abierto. Fue diseñado para dispositivos como teléfonos o tabletas inteligentes, aunque hoy en día también lo encontramos en otro tipo de dispositivos como pueden ser los relojes inteligentes, televisiones o incluso en aparatos electrónicos dentro de los coches.

Gracias a ello, Android ocupa hoy en día el primer puesto dentro de este campo, y una de sus principales razones es por la gran variedad de aplicaciones que proporciona y su uso sencillo.

1.2.1 HISTORIA: ASÍ NACE ANDROID

Android Inc., empresa fundada en 2003, fue la que permitió el nacimiento de este sistema operativo, el cual lleva su nombre. Su objetivo inicial era utilizar

este sistema para enlazar cámaras digitales al PC sin el uso de cables, pero terminaron llevándolo a la telefonía móvil.

- En 2005, Google compra esta compañía por el valor de 50 millones de dólares.
- En 2007 y 2008, es anunciada la primera versión de este sistema, conocido como Android 1.0 Apple Pie y surgen los primeros dispositivos con él.
- En 2010, Android consigue casi la mitad del mercado en Estados Unidos.
- En 2011, el sistema logra liderar el mercado a nivel mundial, superando al sistema iOS, de Apple.

Desde entonces, Android ha ido actualizando constantemente su sistema para poder cubrir la demanda, la cual cada vez varía más rápido y es más exigente. Para ello, han tenido que recurrir a novedades electrónicas y digitales, mejoras de rendimiento, sistemas que incrementan la seguridad de los clientes y actualizaciones constantes de las utilidades proporcionadas al usuario, como son las aplicaciones.

Así es como, Google Play (tienda oficial de aplicaciones de Android) cuenta con más de 3 millones de aplicaciones dentro de este sistema operativo hoy en día.

1.2.2 VERSIONES DEL SISTEMA

Desde que salió la primera versión de Android que, como ya he comentado anteriormente, se identificó como Android 1.0 Apple Pie, este sistema ha ido actualizándose constantemente, siendo todas sus actualizaciones hasta el día de hoy las siguientes:

Ilustración 1: Logo Android Apple Pie

- **Android Apple Pie**: versión 1.0 y fecha de lanzamiento 23 de septiembre de 2008. Sus características más

destacadas son: soporte para wifi y Bluetooth, aplicaciones sencillas, sincronización de Google Contacts, Google Calendar y Gmail, Google Maps, etc. Nivel de API (véase el primer párrafo de la página 13)*: 1.

- **Android Banana Bread**: versión 1.1 y fecha de lanzamiento 9 de febrero de 2009. Sus características más destacadas son: reseñas en Mapas, posibilidad de guardar los archivos adjuntos en mensajes, diseño del sistema, etc. Nivel de API: 2.

Ilustración 2: Logo Android Banana Bread

- **Android Cupcake**: versión 1.5 y fecha de lanzamiento entre el 25 y 30 de abril de 2009. Sus características más destacadas son: soporte para teclados virtuales de terceros, predicción de texto, soporte para Widgets, opciones de copiar y pegar dentro del navegador web, opción de auto-rotación de la pantalla, se permite la subida de videos a YouTube, etc. Nivel de API: 3.

Ilustración 3: Logo Android Cupcake

- **Android Donut**: versión 1.6 y fecha de lanzamiento 15 de septiembre de 2009. Sus características más destacadas son: síntesis de habla para permitir a cualquier aplicación dictar un texto que se encuentra dentro de ella, mejor integración de cámara y galería, selección múltiple dentro de la galería, etc. Nivel de API: 4.

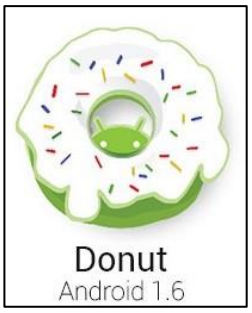

Ilustración 4: Logo Android Donut

- **Android Eclair**: versión 2.0-2.1 y fecha de lanzamiento 26 de octubre de 2009. Sus características más destacadas son: se permite agregar y sincronizar varias cuentas al dispositivo, soporte Bluetooth 2.1, nuevas características para la cámara, mayor velocidad de tecleo en el teclado,

interfaz de usuario de navegar renovada, optimización en velocidad de hardware, posibilidad de fondos de pantalla animados, etc. Nivel de API: 5-7.

Ilustración 5: Logo Android Eclair

- **Android Froyo**: versión 2.2-2.3 y fecha de lanzamiento 20 de mayo de 2010. Sus características más destacadas son: optimización de memoria, velocidad y rendimiento, se habilitan las notificaciones push, deshabilitar acceso de datos sobre red móvil, cambio rápido de lenguaje en el teclado, soporte para Adobe Flash, arreglo de errores varios, etc. Nivel de API: 8.

Ilustración 6: Logo Android Froyo

- **Android Gingerbread**: versión 2.3-2.7 y fecha de lanzamiento 6 de diciembre de 2010. Sus características principales son: soporte para mayor calidad de pantalla, tecnología NFC, nuevos efectos de audio, soporte múltiple para varias cámaras en un mismo dispositivo, mejoras en la batería, etc. Nivel de API: 9-10.

Ilustración 7: Logo Android Gingerbread

- **Android Honeycomb**: versión 3.0-3.2.6 y fecha de lanzamiento 22 de febrero de 2011. Fue la primera actualización exclusiva para televisiones y tabletas y sus características principales son: ‘’modo incógnito’’ que permite la navegación anónima, multitarea simplificada, interfaz simplificada e intuitiva, soporte para procesadores multi-núcleo, videochat mediante Google Talk, conectividad para USB, etc. Nivel de API: 11-13.

Ilustración 8: Logo Android Honeycomb

- **Android Ice Cream Sandwich**: versión 4.0-4.0.5 y fecha de lanzamiento 12 de octubre de 2011. Sus características principales son: creación de carpetas con el método arrastrar/soltar, captura de pantalla integrada, desbloqueo facial, editor de fotos integrado, grabación de video a 1080p, mejoras en gráficos y optimizaciones, etc. Nivel de API: 14-15.

Ilustración 9: Logo Android Ice Cream Sandwich

- **Android Jelly Bean**: versión 4.1-4.3.1 y fecha de lanzamiento 9 de julio de 2012. Sus características principales son: interfaz de usuario más fluida y novedosa, notificaciones expandibles, posibilidad de desactivar las notificaciones de una aplicación específica, reorganización automática de widgets, dictado de voz offline, adicción de Google Wallet, arreglo de fallos, fotos panorámicas, permite envíos directos a impresoras, etc. Nivel de API: 16-18.

Ilustración 10: Logo Android Jelly Bean

Ilustración 11: Logo Android KitKat

- **Android KitKat**: versión 4.4-4.4.4 y fecha de lanzamiento 31 de octubre de 2012. Sus características principales son: barra de navegación oculta en ciertas aplicaciones, nuevos marcos de transiciones y efectos, aumento del límite de sincronización por Bluetooth, acceso directo a fotos desde la cámara, arreglo de diferentes fallos, etc. Nivel de API: 19-20.

Ilustración 12: Logo Android Lollipop

- **Android Lollipop**: versión 5.0-5.1.1 y fecha de lanzamiento 3 de noviembre de 2014. Sus características principales son: diseño totalmente novedoso y colorido, nuevas formas de control de notificaciones, responder mensajes desde la pantalla de bloqueo, clasificación inteligente de las notificaciones, ahorro de batería mejorado, adición de 15 nuevos idiomas, se agrega la aplicación de linterna, etc. Nivel de API: 21-22.

Ilustración 13: Logo Android Marshmallow

- **Android Marshmallow**: versión 6.0-6.0.1 y fecha de lanzamiento 5 de octubre de 2015. Sus características principales son: elección de a qué permisos puede acceder cada aplicación, soporte para huellas dactilares, adición de Android Pay, copias de seguridad automáticas, nueva forma de compartir contenido (Direct Share), soporte para tarjetas SD, compatibilidad con lápices Bluetooth, etc. Nivel de API: 23.

Ilustración 14: Logo Android Nougat

- **Android Nougat**: versión 7.0-7.1.2 y fecha de lanzamiento 22 de agosto de 2016. Sus características principales son: opción de multiventana, optimización de memoria RAM, economizador de consumo de datos, filtro azul en visualización nocturna, mejoras de rendimiento, arreglo de fallos, etc. Nivel de API: 24-25.

- **Android Oreo**: versión 8.0-8.1 y fecha de lanzamiento 21 de agosto de 2017. Sus principales características son: posibilidad de posponer notificaciones y elegir el tiempo en el que queremos recibirla, activación

automática del WiFi al llegar al hogar, mejoras en compatibilidad del sistema, rendimiento y seguridad, rediseño de emojis, etc. Nivel de API: 26-27.

Ilustración 15: Logo Android Oreo

- **Android Pie**: versión 9.0 y fecha de lanzamiento 6 de agosto de 2018. Sus características principales son: sustitución de los botones home, volver y multitarea por una barra de gestos, posibilidad de poner un límite de usos de las aplicaciones, mejoras de batería, rendimiento y seguridad, etc. Nivel de API: 28.

Ilustración 16: Logo Android Pie

- **Android 10 y 11**: versiones 10.0 y 11.0, lanzadas respectivamente el 3 de septiembre de 2019 y el 8 de septiembre de 2020. Sus características principales son: adición de un tema oscuro para todo el sistema y su uso será opcional para el usuario, posible bloqueo de las ranuras SIM, conexión del móvil a una pantalla secundaria y que aparezca con una interfaz similar a la de Windows, introducción del modo ‘’No molestar’’ personalizable, dos modos de rendimiento, reconocimiento facial 3D mejorado, adición de la función Nearby Sharing, etc.

Está previsto que a lo largo del año 2021 salga a la luz la nueva actualización del sistema, que será reconocida como **Android 12**, hasta el momento solo se han distribuido algunas betas.

Tras conocer todas las actualizaciones que han rodeado al sistema operativo Android, podemos percatarnos de que, durante toda su historia, los nombres que Android ha otorgado a sus actualizaciones hacen referencia a diferentes postres o dulces y, además, han ido siguiendo un orden alfabético.
Como curiosidad, durante la presentación de Android KitKat (Android 4.4), la empresa reconoció que este hecho es un reflejo de la misión que tienen los *smartphones* y las tabletas: endulzar nuestras vidas. No obstante, desde la versión Android 10 en adelante, la compañía ha decidido nombrarlas

únicamente con números, con el fin de mantener los nombres lo más inclusivos y accesibles posible. Esto se debe a que no han sido siempre correctamente entendidos debido al desconocimiento de algunos postres en ciertos países. Sin embargo, esto es únicamente de cara al público ya que, a nivel interno de la empresa, las versiones 10 y 11 son conocidas como *Quince Tart* y *Red Velvet Cake,* respectivamente.

Otro aspecto a destacar de las diferentes versiones de Android y que se ha ido comentando en cada descripción de las mismas, es su nivel de API***** (en inglés *Application Programming Interface*, en español interfaz de programación de aplicaciones). Hablamos de API para hacer referencia al conjunto de funciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones, lo que significa que, a mayor nivel de API, mayor número de ventajas para el desarrollador. Este factor es importante ya que, cuando se programa una aplicación, se ha de conocer el nivel de API para el que podrá ser utilizada, debido a que esto hará que una aplicación pueda ser compatible o no con ciertas versiones Android.

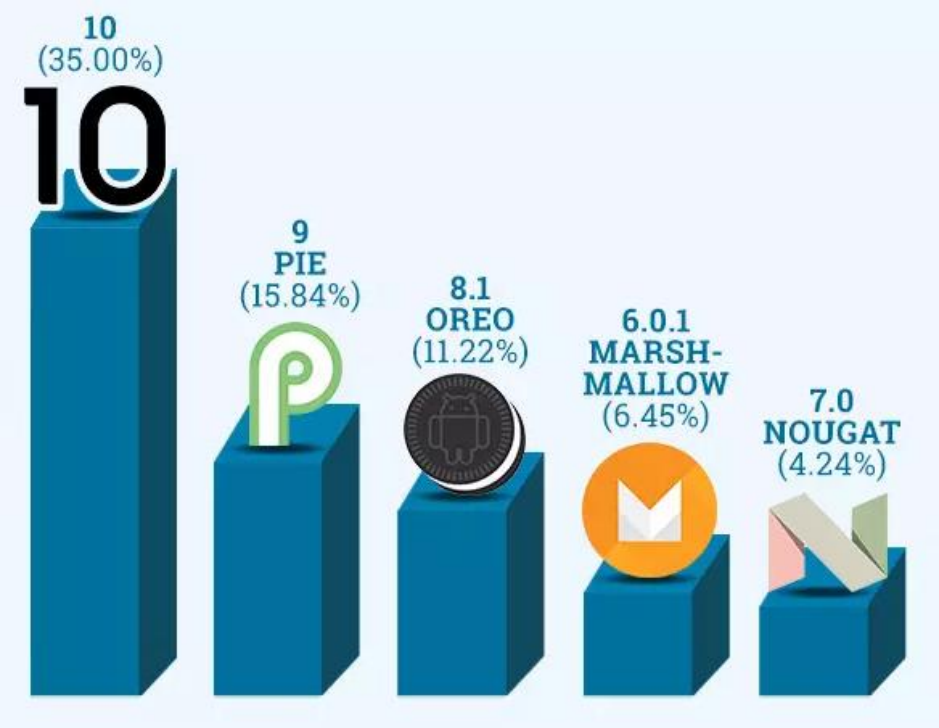

Ilustración 17: Versiones de Android más utilizadas. Marzo 2021.

1.3 HERRAMIENTAS PARA DESARROLLAR LA APLICACIÓN PARA ANDROID

Hay diferentes entornos en los cuales se puede desarrollar una aplicación para el sistema operativo Android. Las principales son: MIT App Inventor 2 y Android Studio.

1.3.1 MIT APP INVENTOR 2

App Inventor es un entorno de desarrollo de software creado por Google para la elaboración de aplicaciones destinadas al sistema operativo de Android. Se caracteriza por su simplicidad y facilidad de uso, ya que es gratuito y fácilmente accesible. Su uso está dirigido a personas que no están muy adentradas en el mundo de la programación informática. Sin embargo, sus aplicaciones están capacitadas para cubrir varias de las necesidades básicas de un dispositivo móvil.

En este entorno, los programas se desarrollan visualmente basándose en la unión o ensamblaje, similar al de unas piezas que se juntan encajando como las de un rompecabezas.

-Funcionamiento básico e interfaz:

1. Mis proyectos → es la pantalla principal de inicio, donde podremos ver todos los proyectos que tengamos guardados o crear uno nuevo.

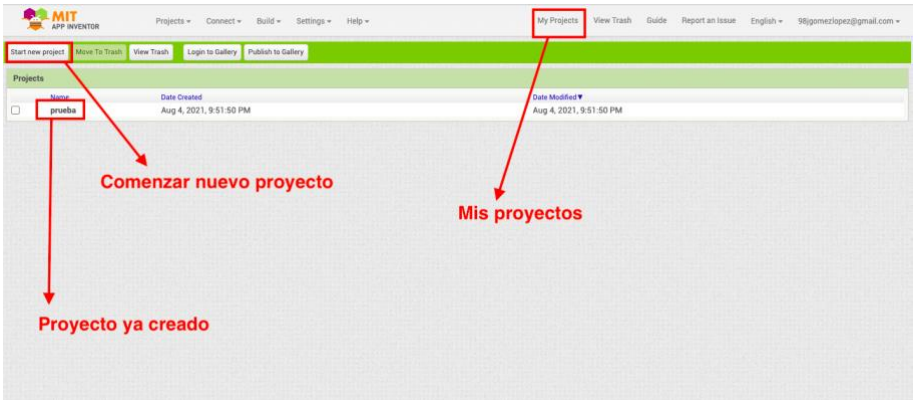

Ilustración 18: Interfaz ‘‘Mis proyectos’’. MIT App Inventor 2.

2. Pantalla principal de la aplicación → una vez entremos a nuestra aplicación, nos encontraremos 2 opciones:

    - Diseñador: donde nos encargaremos de toda la parte de diseño de la aplicación, eligiendo los componentes que irán dentro de ella. Dentro de esta opción, aparecen varias pestañas:

        - Paleta: situada en la parte de la izquierda de la pantalla. Aquí encontramos los componentes que vamos a ir añadiendo a nuestra aplicación. Por ejemplo: interfaz de usuario (botones, casillas de selección, adición de imágenes, desplegables, etc.), *layout* (donde elegiremos si queremos bloques en la izquierda, derecha, en vertical, etc.), y más tipos de componentes.
        - Visor: está situado en la parte central de la ventana. Es donde iremos arrastrando todos los componentes que elijamos de la paleta y podremos ver cómo se compone la aplicación que estemos creando.
        - Componentes: se encuentra a la derecha del visor. Aquí aparecen todos los componentes organizados que van a interactuar con la aplicación.
        - Media: situada debajo de la pestaña de ‘componentes’. Aquí añadiremos cualquier archivo (imágenes, sonidos, etc.) que queramos que aparezca en nuestra aplicación.
        - Propiedades: la encontramos en la parte derecha de la pantalla. En esta pestaña iremos viendo o editando todas las propiedades de los bloques que hemos ido añadiendo desde la paleta al visor. Para ello, pulsaremos desde la pestaña de ‘componentes’ en el componente del cual queramos ver o editar sus propiedades.

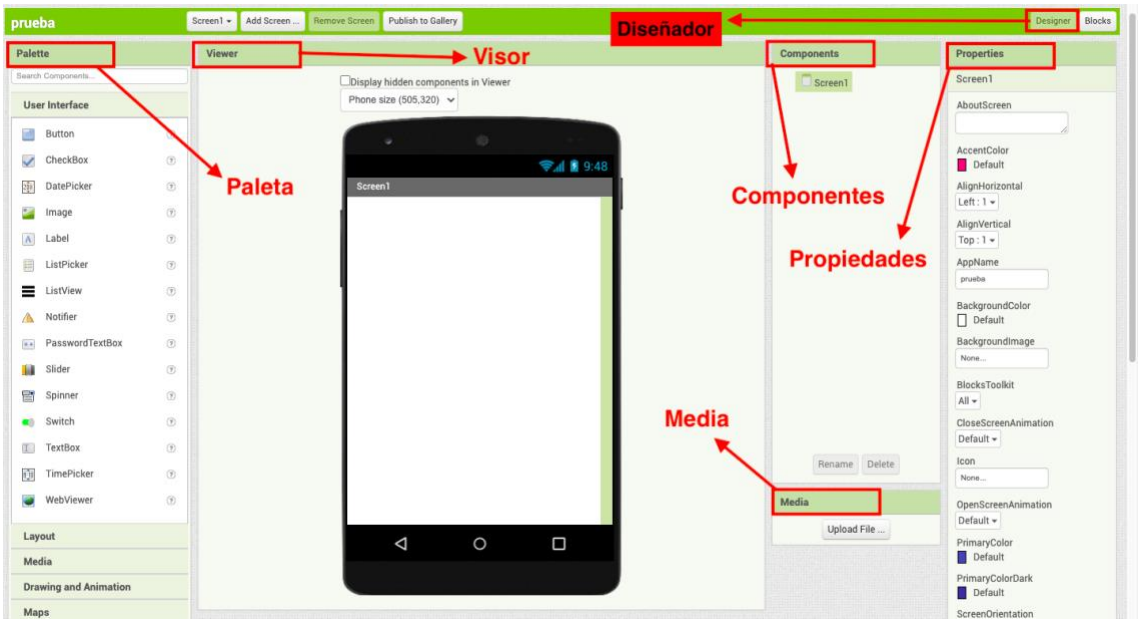

Ilustración 19: Interfaz ‘’Diseñador’’. MIT App Inventor 2.

- Editor de bloques: donde tiene lugar la parte que hace referencia a la programación. Aquí se trabaja con el lenguaje de bloques, los cuales iremos ensamblando para dictar como han de comportarse los componentes dentro de la aplicación.
  Dentro de esta opción, aparecen varias pestañas:

  - Bloques: situado a la izquierda de la ventana. Aquí elegiremos los bloques que queramos agregar (bloques de control, lógicos, matemáticos, etc.) y que irán dictando el comportamiento de los componentes.
  - Media: se encuentra debajo de la pestaña de ‘bloques’. Aquí se añadirán los archivos media deseados.
  - Visor: lo encontramos en la parte central de la ventana. Es donde iremos arrastrando todos los bloques que elijamos de la pestaña ‘bloques’ e iremos ensamblándolos de acuerdo al comportamiento deseado.

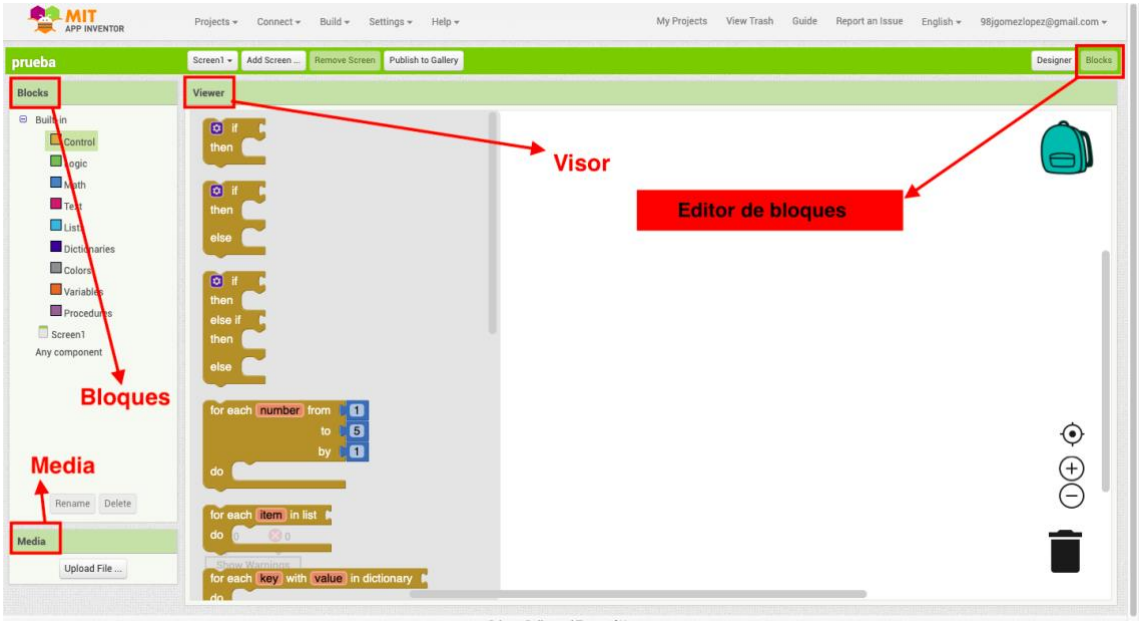

Ilustración 20: Interfaz ''Editor de bloques''. MIT App Inventor 2.

Algunas de las ventajas que presenta son:

- Posibilidad de crear aplicaciones mediante bloques de manera intuitiva y gráfica, sin necesidad de saber código de programación.
- Posibilidad de acceder en cualquier momento y lugar, siempre que haya conexión a internet.
- Varias formas de conectividad: directa, wifi o mediante emulador.
- Facilidad y ahorro de tiempo en comparación con otros entornos.

Por otro lado, contamos con ciertos inconvenientes:

- No existe generación de código Java para desarrollos mas profundos.
- Principalmente, su entorno de desarrollo es Android. No obstante, con el paso del tiempo se ha agregado la opción de desarrollo en iOS, pero ésta no está tan desarrollada como la de Android.

1.3.2 ANDROID STUDIO

Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones para el sistema operativo Android. Anteriormente, el desarrollo de aplicaciones Android se llevaba a cabo mediante el software Eclipse, sin embargo, fue reemplazado por éste en 2013. Por tanto y como ya se ha mencionado anteriormente, es el entorno elegido para el desarrollo de este proyecto.

Está basado en el software IntelliJ IDEA de JetBrains y ha sido publicado de forma gratuita a través de la Licencia Apache 2.0. Además, está disponible para las plataformas Microsoft Windows, macOS, Google Chrome OS y GNU/Linux, y admite varios lenguajes de programación, como Kotlin, Java y C++, por lo que es capaz de alcanzar a un elevado número de usuarios.

-Interfaz general del entorno:

- Administrador de proyectos: está situada a la izquierda de la ventana. En esta pestaña veremos todos los archivos y carpetas que le darán estructura a nuestra aplicación. Cada módulo de la aplicación contendrá las siguientes carpetas:
    - Manifest: contiene el archivo AndroidManifest.xml. Este archivo proporciona información esencial sobre la aplicación al sistema Android, información que el sistema debe tener para poder ejecutar el código.
    - Java: contiene los archivos de código fuente de Java. Cada uno de los archivos que encontramos en esta carpeta corresponde con una de las interfaces de usuario que hemos creado, su función dentro de la aplicación es dotar de funcionalidades a todos los elementos que hemos creado anteriormente, como botones, visores de texto, ventanas editables de texto, etc.
    - Res: contiene todos los recursos como diseños XML, cadenas de IU e imágenes de mapa de bits. Principalmente, se destaca el

Layout, que es donde se encuentra la interfaz de la aplicación basada en archivos xml. Así, accederemos a las funcionalidades de la aplicación interactuando con dicha interfaz, por ello es necesario planificarla correctamente. De esta manera, tendremos una interfaz funcional y sencilla de utilizar.

- Paleta de componentes: cuando trabajamos en modo diseño, esta ventana contiene todos los componentes que podemos colocar en la interfaz de nuestra aplicación.
- Árbol de componentes: es una lista de forma de árbol jerárquico en la que veremos los componentes que hemos colocado en la interfaz de nuestra aplicación y cómo se agrupan.
- Propiedades: en esta pestaña veremos la lista de propiedades que tiene cada uno de los componentes que hemos agregado a la interfaz.

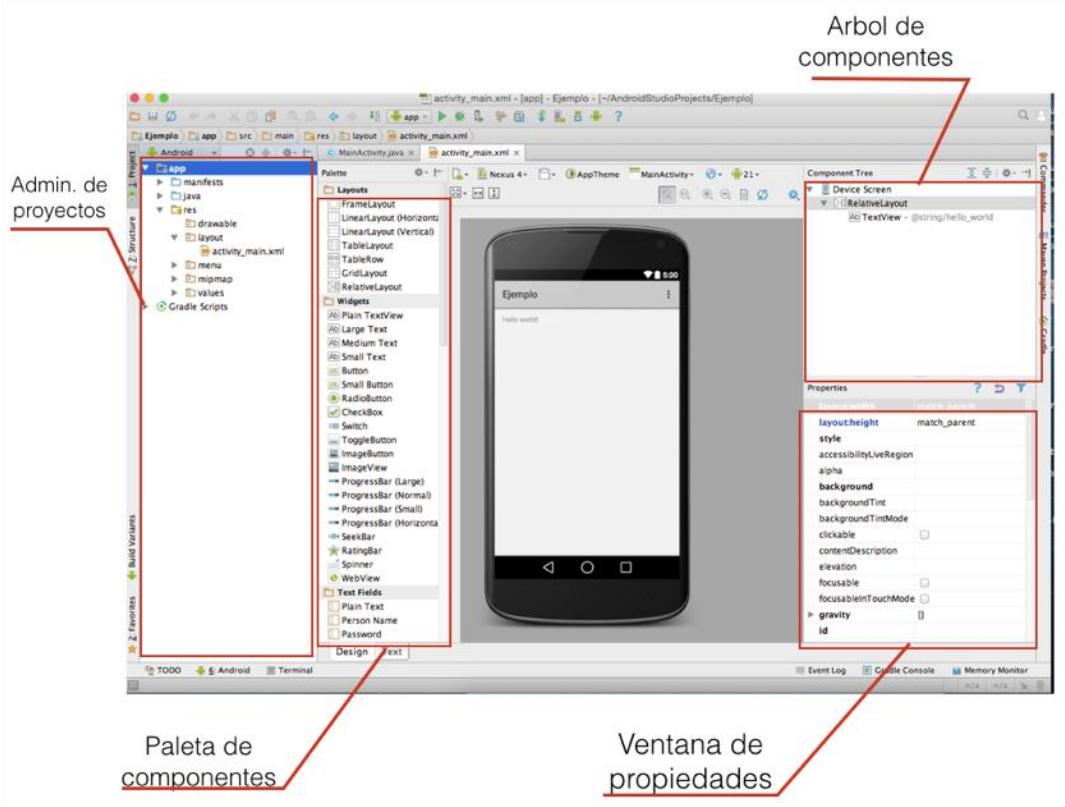

Ilustración 21: Interfaz principal. Android Studio.

Algunas de las ventajas que presenta son:

- Rápida compilación.
- Ejecución de la aplicación a tiempo real y posibilidad de ejecutarla directamente desde el teléfono móvil o tableta.
- Simulación en diferentes dispositivos.
- Capacidad de profundizar correctamente en el desarrollo de una aplicación.
- Permite la creación de nuevos módulos dentro de un mismo proyecto, sin necesidad de estar cambiando de espacio de trabajo para el manejo de proyectos, como pasaba con su antecedente Eclipse.
- La construcción de los paquetes apk, mediante el uso de Gradle facilita la distribución y reutilización del código y recursos, lo que simplifica el desarrollo.

Por otro lado, contamos con ciertos inconvenientes:

- Curva de aprendizaje más compleja.
- Requiere de una gran cantidad de recursos y gasta bastante batería. Así que, para que el emulador trabaje correctamente, se necesita un buen equipo de trabajo, con gran capacidad de RAM y espacio suficiente en el disco duro.

	Windows	OS X/macOS	Linux
OS version	Windows 10/8/7 (32- o 64-bit)	Mac OS X 10.10 (Yosemite) o superior, hasta 10.13 (macOS High Sierra)	GNOME o KDE desktop
RAM	4 GB RAM mínimo, 8 GB RAM recomendado más 1GB adicional para el emulador de Android		
Espacio de almacenamiento	2 GB para Android Studio, 4 GB recomendados (500 MB para IDE y al menos 1.5 GB para Android SDK, imágenes de sistema de emulador y cachés)		
Java version	Java Development Kit (JDK) 8		
Resolución de pantalla	1280x800 mínimo, 1440x900 recomendado		

Ilustración 22: Requisitos recomendados para el uso de Android Studio.

1.4 APLICACIONES SIMILARES

En este apartado se mostrarán algunas aplicaciones con funciones similares a las de la app desarrollada en este proyecto. Éstas han servido de apoyo e inspiración para algunos apartados o características de nuestra aplicación.

Diario de la mascota

En esta aplicación puedes agregar diferentes tipos de mascotas (como por ejemplo perros, gastos, peces o pájaros) y hacer un leve seguimiento de sus horarios.

Cuando creas una mascota nueva, automáticamente se agregan ciertos tipos de alertas, como la alerta de comprar comida o la de salir a pasear. También, puedes agregar alertas para ir a visitar al veterinario. El apartado donde se encuentra esto tiene un símbolo de una campanita.

Además, para agregar características de la mascota, está la opción de añadir texto en el apartado de notas. El apartado donde se encuentra esta opción tiene un símbolo de un lápiz.

Todo esto se muestra en conjunto en modo calendario en la pantalla principal de la mascota.

Esta aplicación tiene la capacidad de realizar una copia de seguridad y compartirla, por lo que los datos no se perderán en caso de que ocurra algún tipo de fallo con la aplicación o el dispositivo. Esta opción se encuentra en la configuración de la aplicación.

A continuación, agrego algunas capturas de pantalla en las cuales se puede apreciar el funcionamiento descrito:

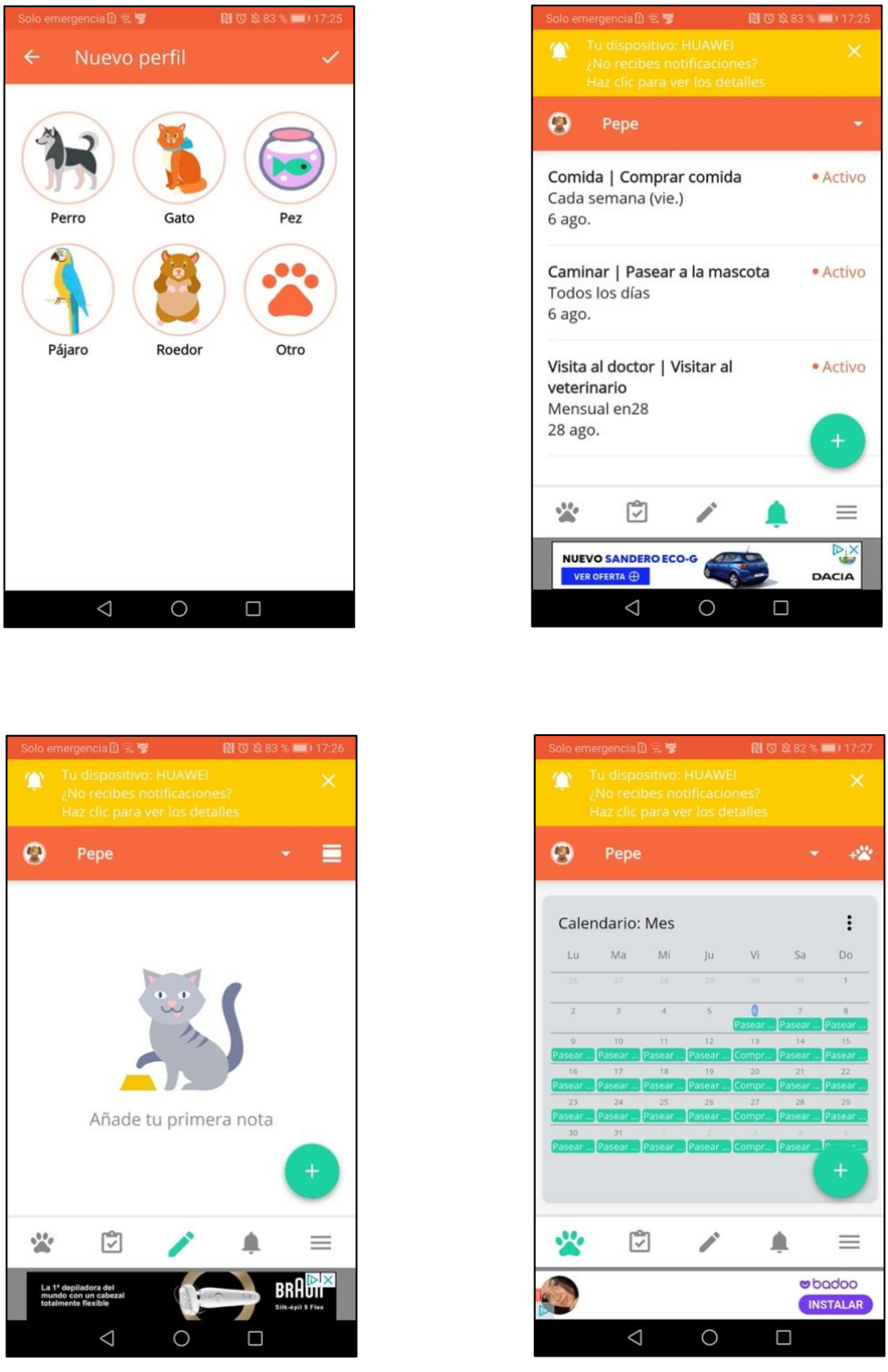

Ilustración 23: Pantallas de la aplicación Diario de la mascota.

Cuidado de mi mascota

Esta aplicación permite agregar diferentes mascotas e información acerca de ellas. Sin embargo, lo especial de esta aplicación es su diseño y sus opciones de personalización, ya que presenta un diseño muy llamativo con diferentes colores, texturas y fondos que son totalmente personalizables.

En la pantalla principal de la aplicación, podemos ver la lista de mascotas agregadas y, en el menú inferior, encontramos otras funciones como la de personalización (donde se nos permite cambiar el tema principal), la tienda (donde se pueden adquirir packs de fondos, *stickers*, íconos de animales nuevos, etc. a cambio de una cantidad de dinero) y los ajustes de la aplicación.

Las mascotas que se podrán agregar inicialmente son: perros, gatos, hámsteres, pájaros, conejos, tortugas, peces y serpientes. El resto de animales a agregar están bloqueados y aparecen con un ícono de un huevo, el cual se desbloqueará (mediante pago o visualización de anuncios) y aparecerá el animal que se esconde tras él.

Dentro de la ficha de cada animal, se introducen algunos datos acerca de éste, los cuales son: el nombre del dueño, el nombre del animal, su apodo, y su género. También hay diferentes apartados dentro de cada ficha:

- Diseño: mediante el cual se puede cambiar el fondo de la ficha de la mascota, el color de los botones y el tipo de mascota seleccionada.
- Información sobre la mascota: donde se puede ver la información que se ha introducido al principio y que ya he mencionado anteriormente (nombre del animal, apodo, etc.).
- Pin: este apartado permite crear una contraseña para acceder a la ficha de la mascota.

A continuación, agrego algunas capturas de pantalla en las cuales se puede apreciar el funcionamiento descrito:

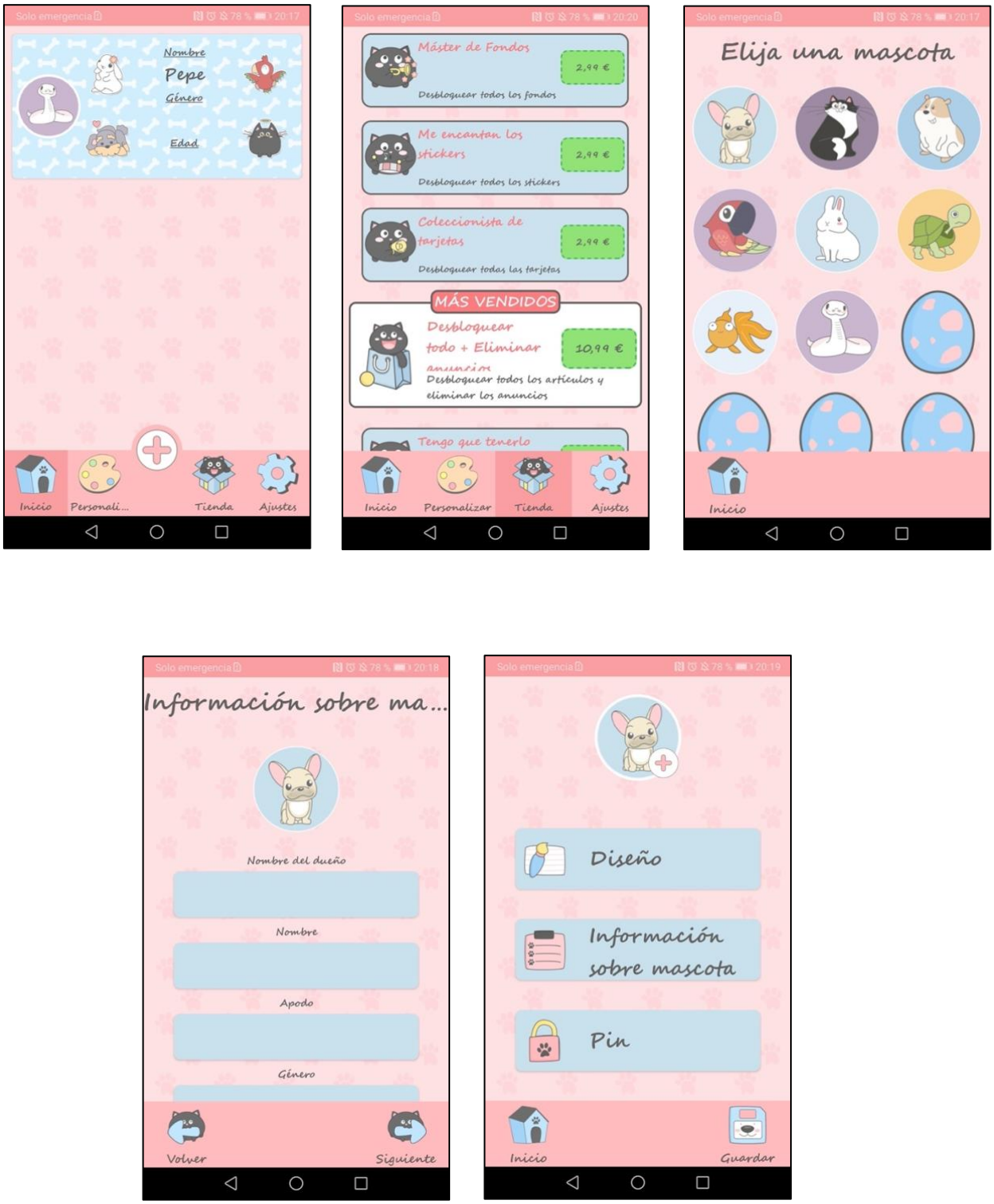

Ilustración 24: Pantallas de la aplicación Cuidado de mi mascota.

1.5 DESCRIPCIÓN DE LA APLICACIÓN

La aplicación se llama DoggyControl y su objetivo es llevar un control adecuado de una o varias mascotas caninas, además de aportar información útil para su dueño/a. Para ello, permite almacenar, de manera eficiente y organizada, diferentes datos de cada animal que se agregue, también se podrá agregar recordatorios para un mejor control de las mascotas y, por si fuera necesario, la aplicación cuenta con un apartado dedicado al contacto con el veterinario.

DoggyControl cuenta con varias actividades que, unidas entre sí, dan sentido a la aplicación. Se ha de saber que cada actividad tiene un archivo .xml asociado, en el cual se construye el diseño o la interfaz de usuario de la actividad, y un archivo .java, que es donde se agrega el código o la programación de dicha actividad para poderla llevar a cabo.

La primera actividad se llama “Main Activity” y es la que nos aparece cuando abrimos la aplicación:

Ilustración 25: Pantalla principal de DoggyControl

Tal y como se ve en la ilustración 25, esta actividad presenta en la parte baja el logo de DoggyControl, y 3 botones, que serán las 3 opciones principales que

tendrá nuestra app.

- El primer botón, **“Mis mascotas’’**: cuando pulsamos este botón se inicia la actividad ‘’Pets Activity”, la cual se encarga del control de la información de las mascotas. Esta pantalla consta de un botón, “+Agregar mascota” (que cuando se pulsa se inicia la actividad “Add Activity”) y de una lista, (donde irán apareciendo las mascotas agregadas).

Ilustración 26: Pantalla de Pets Activity de DoggyControl

Dentro de la actividad “Add Activity”, podremos agregar todos los datos de la mascota: imagen (que se podrá elegir entre una de la galería o tomarla con la cámara en ese momento), nombre, fecha de nacimiento (donde se podrá elegir la fecha directamente desde un calendario que se abrirá en forma de cuadro de diálogo al pulsar en el campo de texto), sexo (se elegirá de un desplegable “Macho” o “Hembra”), peso, número de microchip, raza, pedigree (se elegirá de un desplegable “Sí” o “No”, esto es interesante si el animal se presenta a un concurso oficial de belleza, por ejemplo), propietario/a, alergias, vacunas administradas, tratamientos administrados, desparasitación interna, desparasitación externa y un último apartado donde se podrá agregar libremente

información adicional de la mascota. Finalmente, consta de un botón de “Guardar”, que almacenará la información introducida en la base de datos y se visualizará en la lista de la ilustración 26.

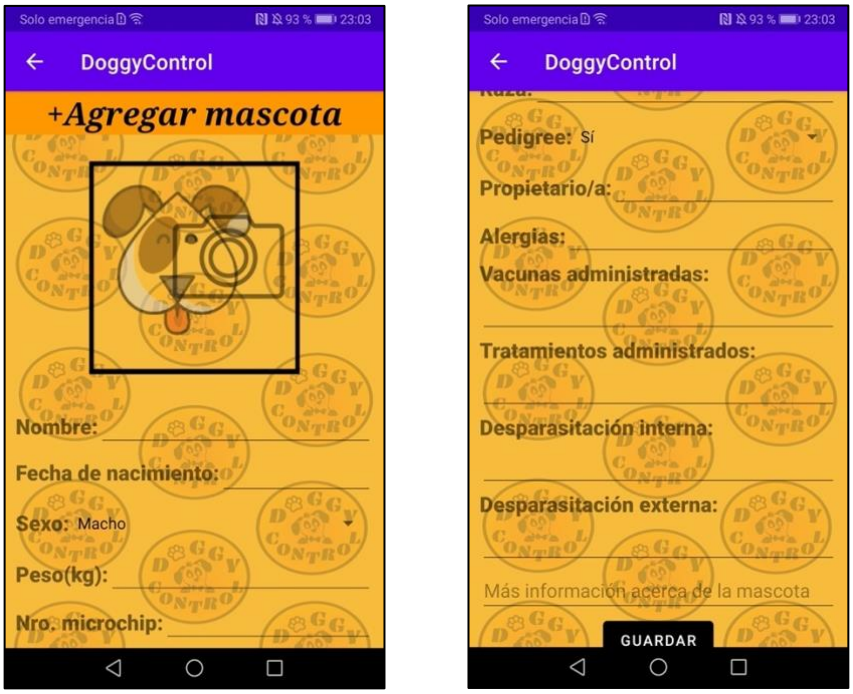

Ilustración 27: Pantalla de Add Activity de DoggyControl

Volviendo a la pantalla de la actividad “Pets Activity” (ilustración 26), cada vez que se pulse en el nombre de alguna mascota que se encuentre en la lista, se iniciará la actividad “Details Activity”, en la cual podremos visualizar todos los datos agregados de la mascota y actualizarlos. Los elementos de esta pantalla son los mismos que la pantalla de “Add Activity”, excepto el botón final, que esta vez cumple la función de “Actualizar”, actualizando así los datos que se hayan modificado de la mascota y refrescando la lista. Además, si se mantiene pulsado el nombre de una mascota unos segundos, el teléfono vibrará y ésta se eliminará de la base de datos y, por tanto, de la lista.

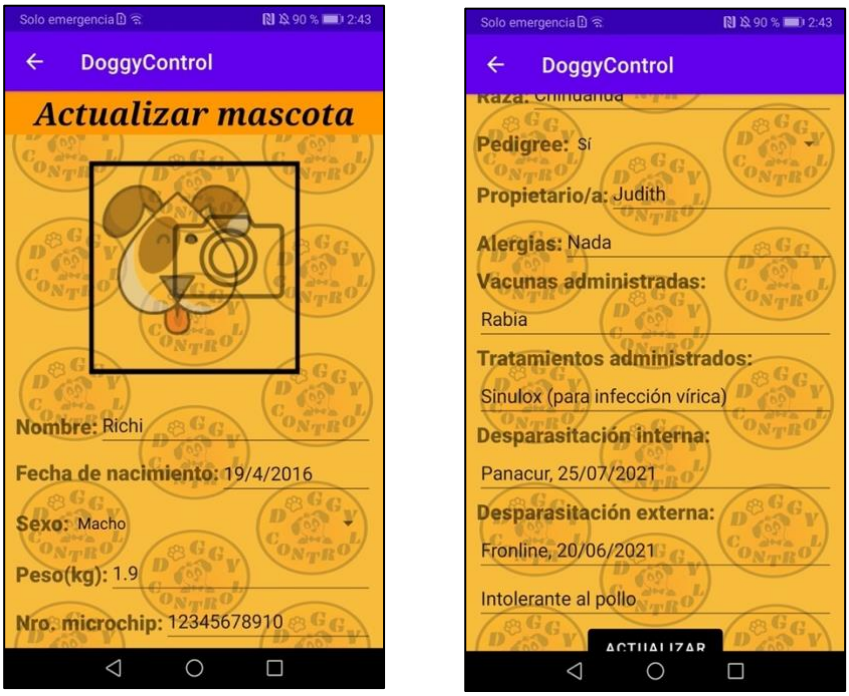

Ilustración 28: Pantalla de Details Activity de DoggyControl

- El segundo botón, **“Información extra”**: al pulsar este botón se inicia la actividad “Inf Activity”, en la cual aparecerá información acerca de las mascotas caninas que podría ser de interés para sus dueños. Esta pantalla consta de dos botones:

  El primero llamado “Recordatorios”, que cuando se pulse se nos abrirá la actividad “Alarma Activity”, donde podremos crear notificaciones insertando un título y una descripción y que serán enviadas en la fecha y hora seleccionadas por el usuario.

  El otro botón es el de “Enfermedades”, al pulsarlo se inicia la actividad “Illness Activity”, la cual muestra un mensaje informativo seguido de 18 botones, cada uno de ellos es una de las enfermedades más comunes en perros. Al pulsar cada uno de estos botones, se abre un cuadro de diálogo con información acerca de dicha enfermedad: descripción de la enfermedad, sus síntomas más comunes y si es o no contagiosa de perros a humanos. Toda esta información está supervisada y contrastada con un veterinario.

En un futuro se podría ir agregando más apartados que contengan información interesante.

Ilustración 29: Pantalla de Inf Activity de DoggyControl

Ilustración 30: Pantalla de Alarma Activity de DoggyControl

Ilustración 31: Pantalla de Illness Activity de DoggyControl

- El tercer botón, **“Contactar veterinario”**: cuando pulsamos este botón, se inicia la actividad “Contact Activity”. Esta actividad se utilizará para ponerse en contacto con el veterinario si en algún momento se requiere. En esta pantalla se muestran 3 botones y un campo de texto para agregar un número de teléfono. El primer botón lanza la aplicación del correo, el segundo botón llama al número que se haya agregado en el campo que se encuentra arriba del mismo, y el tercer botón lanza la aplicación de WhatsApp.

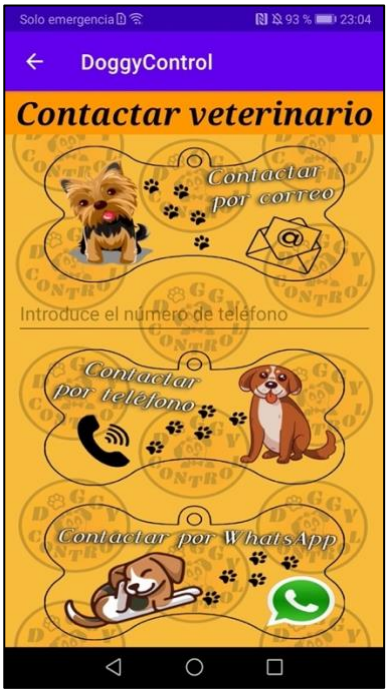

Ilustración 32: Pantalla de Contact Activity de DoggyControl

Todas las actividades presentan arriba a la izquierda una flecha (←) de navegación hacia atrás que, cuando se pulse, dirigirá al usuario a la actividad anterior.

CAPÍTULO 2. MATERIAL Y MÉTODOS

En esta parte del proyecto explicaré las herramientas utilizadas para desarrollar la aplicación: Adobe Photoshop y Android Studio.

2.1 ADOBE PHOTOSHOP

La mayoría de los elementos que vemos en DoggyControl están diseñados y llevados a cabo por mí. Para ello, he recurrido a Adobe Photoshop.

Adobe Photoshop es un editor de fotografías conocido mundialmente. Fue

desarrollado por Adobe Systems Incorporated y se usa principalmente para retocar fotografías y gráficos.
Photoshop puede editar y componer imágenes rasterizadas y soporta varios modelos de colores: RGB, CMYK, CIELAB, colores sólidos y semitonos. Para soportar esto, Photoshop usa sus propios formatos de archivo PSD.

Decidí usar esta herramienta para crear todos mis diseños con el fin de adquirir más conocimientos sobre ella. Una vez que he podido trabajar con Photoshop, puedo afirmar que, al menos para mi objetivo, ha sido el programa de edición más eficaz y versátil.
Los aspectos de este programa que he usado en casi todos mis diseños han sido:

- Edición y eliminación de fondos.
- Superposición y combinación de diferentes imágenes.
- Trabajo con formas.
- Uso y manejo de diferentes tipos de pinceles.
- Trabajo con capas.
- Rasterización de capas para su posterior manejo.
- Personalización completa de textos.
- Trabajo con formato .psd.
- Trabajo con efectos de textura.

Algunos ejemplos son:

Ilustración 33: Logo de DoggyControl

Ilustración 34: Botón de “Recordatorios”

Ilustración 35: Botón de “Contactar por correo”

2.2 ANDROID STUDIO

Antes de empezar a programar nuestra aplicación en esta plataforma, hay que realizar unos cuantos pasos previos:

- Crear un nuevo proyecto de actividad vacía.
- Configurar dicho proyecto.
  Para ello, debemos introducir el nombre de la aplicación (DoggyControl), la localización donde queremos que se guarde, el lenguaje de programación (nuestra aplicación será programada en lenguaje Java) y el nivel de API deseado (elegiremos el recomendado por el sistema, es decir, el nivel de API 16, lo que significa que estará disponible desde la actualización de software Android 4.1 Jelly Bean en adelante, siendo viable para el 99,8% de dispositivos).
- Por último, crear nuestra primera actividad.
  Nuestra actividad principal se llamará MainActivity, que es el nombre que por defecto pone Android Studio.

Una vez realizados estos pasos, podemos empezar a programar nuestra aplicación. Para ello, recurrimos a archivos .xml, donde se programará la interfaz del usuario, y a archivos .java, en los cuales se llevará a cabo toda la programación interna de la aplicación. Éstos se irán presentando y explicando en las futuras páginas.

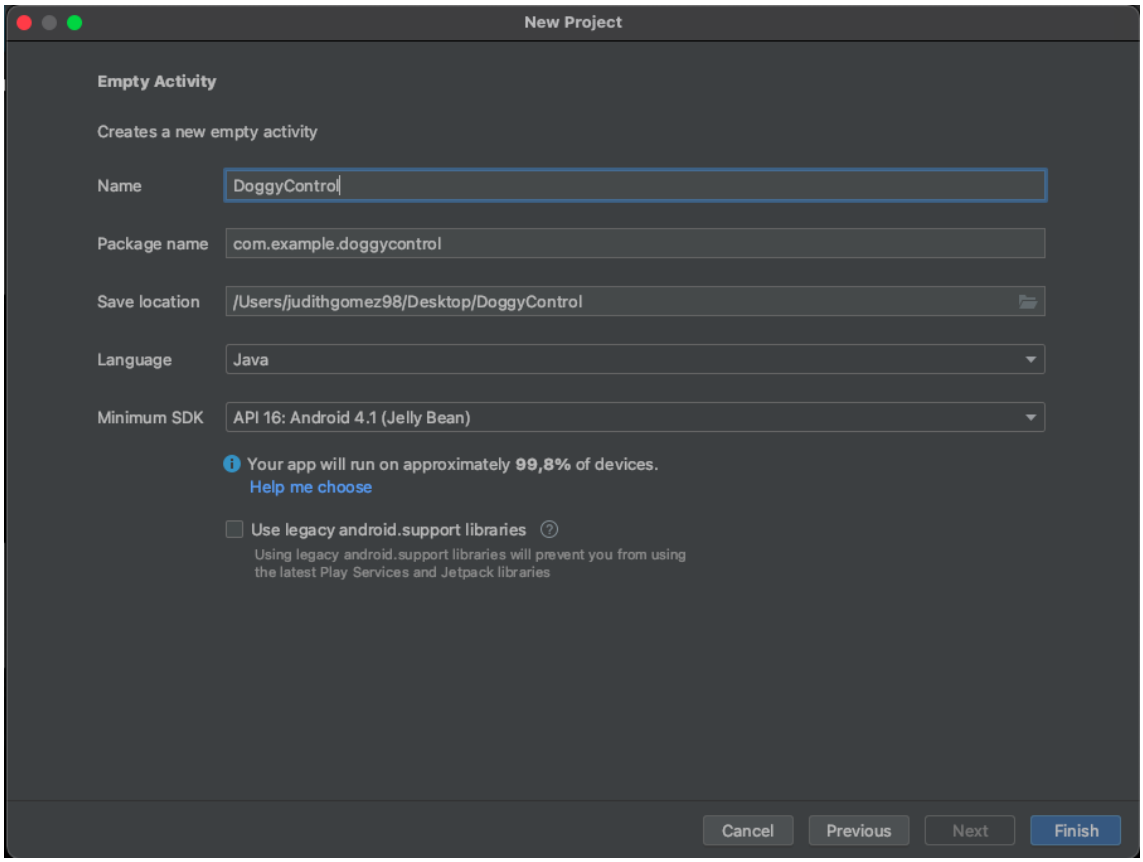

Ilustración 36: Configuración inicial de DoggyControl en Android Studio

2.3 ANDROID MANIFEST

Todas las aplicaciones deben tener un archivo AndroidManifest.xml (con ese nombre exacto) en el directorio raíz. Éste se genera automáticamente al iniciar un nuevo proyecto en Android Studio.

El archivo de manifiesto describe las características fundamentales de la aplicación y de cada uno de sus componentes. Funciona como una interfaz entre el sistema operativo Android y la aplicación, por ello, si no se declaran aquí los componentes de la misma, la aplicación no se ejecutará correctamente.

Entre otras cosas, el archivo de manifiesto hace lo siguiente:

- Nombra el paquete de Java para la aplicación. El nombre del paquete sirve como un identificador único para la aplicación.

- Describe los componentes de la aplicación, como las actividades, los servicios, los receptores de mensajes y los proveedores de contenido que la integran.
- Nombra las clases que implementa cada uno de los componentes y publica sus capacidades, como los mensajes Intent con los que pueden funcionar. Estas declaraciones notifican al sistema Android los componentes y las condiciones para el lanzamiento.
- Determina los procesos que alojan a los componentes de la aplicación. Declara los permisos que debe tener la aplicación para acceder a las partes protegidas de una API e interactuar con otras aplicaciones.
- Declara los permisos que otros deben tener para interactuar con los componentes de la aplicación.
- Enumera las clases Instrumentation que proporcionan un perfil y otra información mientras la aplicación se ejecuta. Estas declaraciones están en el manifiesto solo mientras la aplicación se desarrolla y se quitan antes de la publicación de esta.
- Declara el nivel mínimo de Android API que requiere la aplicación.
- Enumera las bibliotecas con las que debe estar vinculada la aplicación.

El archivo AndroidManifest.xml correspondiente a este proyecto es el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.doggycontrol">


    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.CALL_PHONE" />
    <uses-permission android:name="android.permission.VIBRATE" />

    <uses-feature android:name="android.hardware.camera"
    android:required="false" />



    <application
        android:allowBackup="true"
        android:icon="@drawable/logo"
        android:label="@string/app_name"
        android:roundIcon="@drawable/logo"
        android:supportsRtl="true"
        android:theme="@style/Theme.DoggyControl">
```

```
        <activity android:name=".PetsActivity"
            android:parentActivityName=".MainActivity">
        </activity>

        <activity android:name=".InfActivity"
            android:parentActivityName=".MainActivity">
        </activity>

        <activity android:name=".ContactActivity"
            android:parentActivityName=".MainActivity">
        </activity>

        <activity android:name=".AddActivity"
            android:parentActivityName=".PetsActivity">
        </activity>

        <activity android:name=".IllnessActivity"
            android:parentActivityName=".InfActivity">
        </activity>

        <activity android:name=".DetailsActivity"
            android:parentActivityName=".PetsActivity">
        </activity>

       <activity android:name=".AlarmaActivity"
           android:parentActivityName=".InfActivity">
       </activity>


        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category
       android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>

</manifest>
```

De este código, destacar lo siguiente:

1. `<uses-permission android:name="android.permission.INTERNET"/>`
   Con esta línea se da permiso a la aplicación a obtener acceso a Internet. La aplicación requiere este permiso para lanzar el correo y la aplicación de WhatsApp.

2. `<uses-permission android:name="android.permission.CALL_PHONE" />`
   Permite a la aplicación a acceder a una llamada telefónica.

La aplicación requiere este permiso para llamar al número introducido en la actividad “Contact Activity”.

3. `<uses-permission android:name="android.permission.VIBRATE" />`
   Permite a la aplicación a que acceda a la vibración del dispositivo.
   La aplicación requiere este permiso para vibrar cuando se elimine una mascota de la lista en la actividad “Pets Activity”.
4. ```
   <uses-feature android:name="android.hardware.camera"
       android:required="false" />
   ```
   Con estas líneas se transmite que el tener cámara en el dispositivo no es un requisito fundamental para el funcionamiento de la aplicación.
5. ```
   <activity android:name=".PetsActivity"
             android:parentActivityName=".MainActivity">
   </activity>
   ```
   Todas las actividades presentan estas líneas. La primera se encarga de nombrar la actividad, en este caso “Pets Activity”, para que se pueda acceder a ella. La segunda se encarga de agregar la fecha (←) de navegación hacia la actividad anterior, en este caso “Main Activity”.

2.4 ARCHIVOS XML

Los archivos .xml se encuentran dentro del apartado de layout de la aplicación (app→res→layout).
Como ya hemos comentado anteriormente, en el layout se encuentra la interfaz de la aplicación basada en estos archivos. De esta manera, el usuario accederá a las funcionalidades de la aplicación interactuando con dicha interfaz, por eso es necesario planificarla, para obtener una interfaz funcional y sencilla de utilizar para el usuario.

Mi aplicación consta de los siguientes archivos correspondientes al layout:

1. activity_main.xml
2. activity_pets.xml
3. activity_add.xml
4. activity_details: este archivo no lo comentaré ya que tiene los mismos

elementos que activity_add.xml.

5. activity_inf.xml
6. activity_alarma.xml
7. activity_illness.xml
8. activity_contact.xml

Para que sea más sencillo comprender el código de estos archivos, comentaré a continuación aspectos, funciones o elementos generales que nos iremos encontrando:

-Linear Layout: alinea todos los elementos que incluyamos dentro de éste en una única dirección, de manera vertical u horizontal.
-ScrollView: permite desplazar el contenido del layout a lo largo de la pantalla, cuando sus dimensiones exceden el tamaño de la misma.
-Button: elemento que el usuario puede pulsar para realizar una acción.
-ImageButton: elemento que permite al usuario utilizar una imagen con función de Button, es decir, permite pulsar sobre éste para realizar una acción.
-TextView: elemento que muestra texto al usuario.
-EditText: elemento que el usuario puede modificar editando su contenido, con el fin de actuar como campo de texto.
-ImageView: elemento que sirve para mostrar imágenes al usuario.
-Spinner: elemento que muestra una lista desplegable con varias opciones, permitiendo al usuario seleccionar una de éstas.
-ListView: elemento que actúa como contenedor de vistas y que permite la creación rápida de una lista vertical de elementos en pantalla con un scroll automático que envuelve a la ListView.

Algunos de los atributos que definen a estos elementos son:

-Tamaño: todos los grupos de vistas incluyen un ancho y una altura (layout_width y layout_height), y cada vista debe definirlos.
Tenemos dos opciones diferentes para definirlos: “wrap_content” para indicar a nuestra vista que modifique su tamaño conforme a los requisitos de un elemento en particular escribiendo, y “match_parent” para indicar a nuestra

vista que se agrande tanto como lo permita el layout.
También se puede especificar el ancho y la altura con medidas exactas.
-Orientación: especificar la dirección en la que se desea ordenar los elementos introducidos en el layout. Puede ser vertical u horizontal.
-Gravity: coloca nuestro elemento en el lugar que le queramos asignar dentro de un elemento contenedor de mayor tamaño que el elemento contenido. Podemos indicar si queremos nuestro elemento en el centro del espacio asignado, en la parte superior, inferior, a la derecha, a la izquierda…etc.
-Peso: especifica la cantidad de espacio adicional en el diseño que se asignará a la vista, recurriendo a “layout_weight”.
-Android ID: conecta los elementos de la interfaz con el código java que los dota de utilidad.

Una vez explicada esta información, pasemos a ver cada uno de los archivos xml de nuestra aplicación.

2.4.1 ACTIVITY_MAIN

El archivo activity_main.xml va ascociado a la actividad “Main Activity”, la cual se encarga de la pantalla principal de DoggyControl.

El archivo activity_main.xml es el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@drawable/fondo"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <ImageButton
        android:id="@+id/idmismascotasbutton"
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_gravity="center"
        android:layout_weight="1"
        android:background="@drawable/mismascotasbutton" />
```

```
    <ImageButton
        android:id="@+id/idinformacionbutton"
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:layout_gravity="center"
        android:background="@drawable/informacionextrabutton" />

    <ImageButton
        android:id="@+id/idcontactarbutton"
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:layout_gravity="center"
        android:background="@drawable/contactarveterinariobutton" />

    <ImageView
        android:id="@+id/logocircular"
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_weight="3"
        android:layout_gravity="center"
        android:src="@drawable/logocircular"
        android:scaleType="fitCenter"/>

</LinearLayout>
```

2.4.2 ACTIVITY_PETS

El archivo activity_pets.xml va ascociado a la actividad “Pets Activity”, la cual se encarga del control de las mascotas.

El archivo activity_pets.xml es el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@drawable/fondo2"
    android:orientation="vertical">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:background="#FF9800"
        android:fontFamily="serif"
        android:text="Mis mascotas"
        android:textColor="#000000"
        android:textSize="40sp"
```

```
        android:textAlignment="center"
        android:textStyle="bold|italic" />

    <ImageButton
        android:id="@+id/idagregarmascotabutton"
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:layout_gravity="center"
        android:background="@drawable/agregarmascotabutton" />


    <ListView
        android:id="@+id/lista_mascotas"
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_weight="4" />


</LinearLayout>
```

2.4.3 ACTIVITY_ADD

El archivo activity_add.xml va asociado a la actividad “Add Activity”, la cual se encarga de la inserción de la información y los datos de las mascotas.

El archivo activity_add.xml es el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/activity_add"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="@drawable/fondo2">

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical">


            <TextView
                android:id="@+id/agregartitulo"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
```

```
            android:layout_gravity="center"
            android:background="#FF9800"
            android:fontFamily="serif"
            android:text="+Agregar mascota"
            android:textColor="#000000"
            android:textSize="30sp"
            android:textAlignment="center"
            android:textStyle="bold|italic" />

        <ImageView
            android:id="@+id/idimagen"
            android:layout_width="200dp"
            android:layout_height="250dp"
            android:layout_gravity="center"
            app:srcCompat="@drawable/addphoto"
            android:scaleType="fitCenter" />

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal">

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginTop="10dp"
                android:layout_marginLeft="10dp"
                android:fontFamily="sans-serif-black"
                android:text="Nombre:"
                android:textSize="20sp"
                android:textStyle="bold" />

            <EditText
                android:id="@+id/idname"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_gravity="center"
                android:layout_marginRight="10dp"/>

        </LinearLayout>

        <LinearLayout
               android:layout_width="match_parent"
               android:layout_height="wrap_content"
               android:orientation="horizontal">

                   <TextView
                   android:layout_width="wrap_content"
                   android:layout_height="wrap_content"
                   android:layout_marginLeft="10dp"
                   android:fontFamily="sans-serif-black"
                   android:text="Fecha de nacimiento:"
                   android:textSize="20sp"
                   android:textStyle="bold" />

                   <EditText
                   android:id="@+id/iddate"
                   android:layout_width="match_parent"
                   android:layout_height="wrap_content"
                   android:layout_marginRight="10dp"
                   android:inputType="date"
```

```
                android:focusable="false"
                android:clickable="true"
                android:maxLines="1" />

        </LinearLayout>

        <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:orientation="horizontal">

                <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginTop="10dp"
                android:layout_marginLeft="10dp"
                android:fontFamily="sans-serif-black"
                android:text="Sexo:"
                android:textSize="20sp"
                android:textStyle="bold" />

                <Spinner
                android:id="@+id/idgender"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_gravity="center"
                android:layout_marginRight="10dp" />

        </LinearLayout>


        <LinearLayout
              android:layout_width="match_parent"
              android:layout_height="wrap_content"
              android:orientation="horizontal">

                <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginTop="10dp"
                android:layout_marginLeft="10dp"
                android:fontFamily="sans-serif-black"
                android:text="Peso(kg):"
                android:textSize="20sp"
                android:textStyle="bold" />

                <EditText
                android:id="@+id/idpeso"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_gravity="center"
                android:inputType="numberDecimal"
                android:layout_marginRight="10dp"/>

        </LinearLayout>


        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal">
```

```
            <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp"
            android:layout_marginLeft="10dp"
            android:fontFamily="sans-serif-black"
            android:text="Nro. microchip:"
            android:textSize="20sp"
            android:textStyle="bold" />

            <EditText
            android:id="@+id/idchip"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:inputType="number"
            android:layout_marginRight="10dp" />

    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

            <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp"
            android:layout_marginLeft="10dp"
            android:fontFamily="sans-serif-black"
            android:text="Raza:"
            android:textSize="20sp"
            android:textStyle="bold" />

            <EditText
            android:id="@+id/idraza"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:layout_marginRight="10dp" />

    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

            <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp"
            android:layout_marginLeft="10dp"
            android:fontFamily="sans-serif-black"
            android:text="Pedigree:"
            android:textSize="20sp"
            android:textStyle="bold" />

            <Spinner
```

```
            android:id="@+id/idpedigree"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:layout_marginRight="10dp"/>

    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

            <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp"
            android:layout_marginLeft="10dp"
            android:fontFamily="sans-serif-black"
            android:text="Propietario/a:"
            android:textSize="20sp"
            android:textStyle="bold" />

            <EditText
            android:id="@+id/idpropiet"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:layout_marginRight="10dp"/>

    </LinearLayout>


    <LinearLayout
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:orientation="horizontal">

            <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp"
            android:layout_marginLeft="10dp"
            android:fontFamily="sans-serif-black"
            android:text="Alergias:"
            android:textSize="20sp"
            android:textStyle="bold" />

            <EditText
            android:id="@+id/idalergias"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:layout_marginRight="10dp" />

    </LinearLayout>


       <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
 android:layout_marginLeft="10dp"
 android:fontFamily="sans-serif-black"
 android:text="Vacunas administradas:"
 android:textSize="20sp"
 android:textStyle="bold" />

 <EditText
 android:id="@+id/idvacunas"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:layout_gravity="center"
 android:layout_marginLeft="10dp"
 android:layout_marginRight="10dp" />


 <TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginLeft="10dp"
android:fontFamily="sans-serif-black"
android:text="Tratamientos administrados:"
android:textSize="20sp"
android:textStyle="bold" />

<EditText
android:id="@+id/idtratamientos"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:layout_marginLeft="10dp"
android:layout_marginRight="10dp" />


<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginLeft="10dp"
android:fontFamily="sans-serif-black"
android:text="Desparasitación interna:"
android:textSize="20sp"
android:textStyle="bold" />

<EditText
android:id="@+id/iddespint"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:layout_marginLeft="10dp"
android:layout_marginRight="10dp" />


<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginLeft="10dp"
android:fontFamily="sans-serif-black"
android:text="Desparasitación externa:"
```

```
                android:textSize="20sp"
                android:textStyle="bold" />

                <EditText
                android:id="@+id/iddespext"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_gravity="center"
                android:layout_marginLeft="10dp"
                android:layout_marginRight="10dp" />



                <EditText
                android:id="@+id/idmasinf"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:hint="Más información acerca de la mascota"
                android:layout_marginLeft="10dp"
                android:layout_marginRight="10dp"/>

                <Button
                android:id="@+id/idguardar"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="center"
                app:backgroundTint="#000000"
                android:text="Guardar" />

            </LinearLayout>

        </ScrollView>

</LinearLayout>
```

De este código, destacar lo siguiente:

-Dentro del ScrollView se ha de agregar un LinearLayout que contenga todos los elementos que queremos que aparezcan dentro del scroll.
-El LinearLayout general es de orientación vertical, por tanto, para los elementos que se encuentran situados horizontalmente, se crean LinearLayouts específicos en orientación horizontal. Podría haberse utilizado RelativeLayout, pero esta forma de hacerlo me agrada más.
-En el último EditText aparece el atributo “android:hint”, este atributo permite que se muestre al usuario texto dentro del edittext. Sin embargo, cuando el usuario agrega texto por su cuenta, el texto que aparecía en el campo desaparece y se sustituye por el agregado por el usuario.

2.4.4 ACTIVITY_DETAILS

El archivo activity_details.xml va asociado a la actividad “Details Activity”, la cual se encarga de la visualización y edición de la información y los datos de las mascotas agregados en la actividad “Add Activity”.

Como ya he comentado anteriormente, el archivo activity_details.xml no se comentará ya que tiene los mismos elementos que activity_add.xml (comentado en el apartado anterior), excepto el botón de “guardar” que se cambia por el botón “actualizar”.

Se propondrá, como trabajo futuro, crear un único archivo xml para ambas funciones (añadir mascota y editar/actualizar mascota). Ya que, de esta manera, los cambios futuros se podrían realizar en el mismo archivo en el que se agregan los datos iniciales de las mascotas.

2.4.5 ACTIVITY_INF

El archivo activity_inf.xml va asociado a la actividad “Inf Activity”, la cual se encarga del apartado de información de la aplicación.

El archivo activity_inf.xml es el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@drawable/fondo2"
    android:orientation="vertical">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:background="#FF9800"
        android:fontFamily="serif"
        android:text="Información extra"
        android:textColor="#000000"
```

```
        android:textSize="30sp"
        android:textAlignment="center"
        android:textStyle="bold|italic" />

    <ImageButton
        android:id="@+id/idrecordatoriosbutton"
        android:layout_width="wrap_content"
        android:layout_marginTop="5dp"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:layout_gravity="center"
        android:background="@drawable/recordatoriosbutton" />

    <ImageButton
        android:id="@+id/idenfermedadesbutton"
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:layout_gravity="center"
        android:background="@drawable/enfermedadesbutton" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_weight="3"
        android:text="Cosas que se podrían ir agregando en un futuro"
        android:textSize="40sp" />

</LinearLayout>
```

De este código, destacar lo siguiente:

-El último TextView de esta pantalla es únicamente informativo, ya que su fin es transmitir que el espacio sobrante de la interfaz puede ser rellenado en un futuro con aspectos importantes para los usuarios.

2.4.6 ACTIVITY_ALARMA

El archivo activity_alarma.xml va asociado a la actividad “Alarma Activity”, la cual permite al usuario crear los recordatorios que desee recibir.

El archivo activity_alarma.xml es el siguiente:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_alarma"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
```

```
    android:background="@drawable/fondo2">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:background="#FF9800"
        android:fontFamily="serif"
        android:text="+Agregar recordatorio"
        android:textColor="#000000"
        android:textSize="30sp"
        android:textAlignment="center"
        android:textStyle="bold|italic" />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:id="@+id/idtitulo"
        android:hint="Titulo"/>

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:id="@+id/iddescripcion"
        android:hint="Descripción"/>

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textPersonName"
        android:layout_margin="10dp"
        android:id="@+id/iddate2"
        android:hint="Seleccione la fecha"/>

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textPersonName"
        android:layout_margin="10dp"
        android:id="@+id/idhour2"
        android:backgroundTint="#000000"
        android:hint="Seleccione la hora"/>

    <Button
        android:text="Guardar recordatorio"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:id="@+id/btn_guardar"
        android:backgroundTint="#000000" />

</LinearLayout>
```

2.4.7 ACTIVITY_ILLNESS

El archivo activity_illness.xml va asociado a la actividad “Illness Activity”, la cual muestra al usuario información acerca de las enfermedades más comunes que existen en el mundo canino.

El archivo activity_illness.xml es el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
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"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@drawable/fondo2"
    android:orientation="vertical">

    <ScrollView
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

      <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:orientation="vertical">

        <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:background="#FF9800"
        android:fontFamily="serif"
        android:text="Enfermedades"
        android:textColor="#000000"
        android:textSize="40sp"
        android:textAlignment="center"
        android:textStyle="bold|italic" />


        <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="10dp"
        android:layout_marginBottom="10dp"
        android:layout_marginRight="10dp"
        android:layout_marginLeft="10dp"
        android:background="#FFFFFF"
        android:fontFamily="monospace"
        android:text="Los perros pueden sufrir diversas enfermedades a
lo largo de su vida.
El conocimiento de los síntomas y la observación y cuidado de tu perro
pueden ayudar a prevenir muchas de ellas.
No descartes ningún método de prevención y administra a tu perro los
```

```
tratamientos adecuados.
Acude a tu veterinario, este profesional es el más cualificado para
ayudarte a tener un perro sano."
        android:textAlignment="center"
        android:textColor="#000000"
        android:textSize="15sp" />

        <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Borreliosis canina o enfermedad de Lyme"
        android:textColor="#000000"
        android:onClick="ShowDialog1"
        app:backgroundTint="#FFE082" />

        <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Brucelosis"
        android:textColor="#000000"
        android:onClick="ShowDialog2"
        app:backgroundTint="#FFC107" />

        <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Cáncer"
        android:textColor="#000000"
        android:onClick="ShowDialog3"
        app:backgroundTint="#FFE082" />

        <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Conjuntivitis"
        android:textColor="#000000"
        android:onClick="ShowDialog4"
        app:backgroundTint="#FFC107" />

        <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Gastritis aguda (vómito agudo)"
        android:textColor="#000000"
        android:onClick="ShowDialog5"
        app:backgroundTint="#FFE082" />

        <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Hepatitis canina"
        android:textColor="#000000"
        android:onClick="ShowDialog6"
        app:backgroundTint="#FFC107" />
```

```
        <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Leishmaniosis"
        android:textColor="#000000"
        android:onClick="ShowDialog7"
        app:backgroundTint="#FFE082" />

        <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Leptospirosis"
        android:textColor="#000000"
        android:onClick="ShowDialog8"
        app:backgroundTint="#FFC107" />

        <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Mastitis"
        android:textColor="#000000"
        android:onClick="ShowDialog9"
        app:backgroundTint="#FFE082" />

        <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Moquillo"
        android:textColor="#000000"
        android:onClick="ShowDialog10"
        app:backgroundTint="#FFC107" />

        <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Otitis"
        android:textColor="#000000"
        android:onClick="ShowDialog11"
        app:backgroundTint="#FFE082" />

        <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Parvovirus"
        android:textColor="#000000"
        android:onClick="ShowDialog12"
        app:backgroundTint="#FFC107" />

        <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Rabia"
        android:textColor="#000000"
        android:onClick="ShowDialog13"
```

```
        app:backgroundTint="#FFE082" />

        <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Sarna demodécica"
        android:textColor="#000000"
        android:onClick="ShowDialog14"
        app:backgroundTint="#FFC107" />

        <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Sarna sarcóptica o sarna común"
        android:textColor="#000000"
        android:onClick="ShowDialog15"
        app:backgroundTint="#FFE082" />

        <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
       android:text="Torsión o dilatación gástrica (vólvulo gástrico)"
        android:textColor="#000000"
        android:onClick="ShowDialog16"
        app:backgroundTint="#FFC107" />

        <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Toxoplasmosis"
        android:textColor="#000000"
        android:onClick="ShowDialog17"
        app:backgroundTint="#FFE082" />

        <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
     android:text="Traqueobronquitis infecciosa o tos de las perreras"
        android:textColor="#000000"
        android:onClick="ShowDialog18"
        app:backgroundTint="#FFC107" />

        </LinearLayout>

    </ScrollView>

</LinearLayout>
```

2.4.8 ACTIVITY_CONTACT

El archivo activity_contact.xml va asociado a la actividad “Contact Activity”, la cual es la encargada de permitir al usuario contactar con su veterinario, si así lo desea en algún momento.

El archivo activity_contact.xml es el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@drawable/fondo2"
    android:orientation="vertical">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:background="#FF9800"
        android:fontFamily="serif"
        android:text="Contactar veterinario"
        android:textColor="#000000"
        android:textSize="30sp"
        android:textAlignment="center"
        android:textStyle="bold|italic" />

    <ImageButton
        android:id="@+id/idcorreobutton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:layout_gravity="center"
        android:background="@drawable/correobutton"
        android:layout_marginLeft="30dp"
        android:layout_marginRight="30dp"/>

    <EditText
        android:id="@+id/idnumero"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:hint="Introduce el número de teléfono"
        android:inputType="phone"
        android:layout_marginLeft="10dp"
        android:layout_marginRight="10dp" />

    <ImageButton
        android:id="@+id/idtelefonobutton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:layout_gravity="center"
```

```
        android:background="@drawable/telefonobutton"
        android:layout_marginLeft="30dp"
        android:layout_marginRight="30dp"/>

    <ImageButton
        android:id="@+id/idwhatsappbutton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:layout_gravity="center"
        android:background="@drawable/whatsappbutton"
        android:layout_marginLeft="30dp"
        android:layout_marginRight="30dp"/>

</LinearLayout>
```

2.5 ARCHIVOS JAVA

Los archivos .java que completan nuestra aplicación se encuentran dentro del apartado de com.example.doggycontrol de la aplicación (app→java→ com.example.doggycontrol).
Nos encontramos con 11 archivos .java, de los cuales 8 se corresponden con una de las interfaces xml comentadas anteriormente. Los otros 3 restantes ayudan a completar la aplicación para que cumpla con su funcionamiento correctamente. Éstos últimos también los iremos comentando en las futuras páginas.

Mi aplicación consta de los siguientes archivos correspondientes a la programación java:

1. MainActivity.java
2. DBHelper
3. AddActivity.java
4. BeanMascotas.java
5. PetsActivity.java
6. DetailsActivity.java

7. InfActivity.java
8. AlarmaActivity.java
9. Workmanagernoti.java
10. IllnessActivity.java
11. ContactActivity.java

He elegido este orden para explicarlo ya que:

1- Antes de explicar la actividad AddActivity.java, se ha de conocer DBHelper.java, que es donde se crea la tabla de la base de datos y la consulta preparada para insertar mascotas que veremos más adelante.
2- Antes de explicar la actividad PetsActivity.java, se ha de conocer BeanMascotas.java, que es el bean que nos ayudará a llevar los datos de la base de datos al ListView de esta actividad.
3- Una vez conocidos los archivos DBHelper.java y BeanMascotas.java, podemos pasar a analizar el resto de archivos de este apartado: PetsActivity.java y DetailsActivity.java.
4- Antes de explicar la actividad AlarmaActivity.java, se ha de conocer Workmanagernoti.java, que es la clase que se encargará de enviar las notificaciones.
5- Una vez conocido el archivo Workmanagernoti.java, podemos pasar a analizar la actividad AlarmaActivity.java.

2.5.1 MAINACTIVITY

El archivo MainActivity.java es el que complementa al archivo activity_main.xml.

El archivo MainActivity.java es el siguiente:

```
package com.example.doggycontrol;

import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
```

```
import android.widget.ImageButton;

public class MainActivity extends AppCompatActivity {

    ImageButton mismascotasboton;
    ImageButton informacionboton;
    ImageButton contactarboton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mismascotasboton = (ImageButton)
findViewById(R.id.idmismascotasbutton);
        informacionboton = (ImageButton)
findViewById(R.id.idinformacionbutton);
        contactarboton = (ImageButton)
findViewById(R.id.idcontactarbutton);


        mismascotasboton.setOnClickListener(new View.OnClickListener()
{
            @Override
            public void onClick(View view) {
                Intent gotopets = new Intent(MainActivity.this,
PetsActivity.class);

                startActivity(gotopets);
            }
        });


        informacionboton.setOnClickListener(new View.OnClickListener()
{
            @Override
            public void onClick(View view) {
                Intent gotoinf = new Intent(MainActivity.this,
InfActivity.class);

                startActivity(gotoinf);
            }
        });


        contactarboton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent gotocontact = new Intent(MainActivity.this,
ContactActivity.class);

                startActivity(gotocontact);
            }
        });

    }
}
```

Lo primero que hacemos en este archivo es definir las variables con las que

vamos a trabajar:

```
ImageButton mismascotasboton;
ImageButton informacionboton;
ImageButton contactarboton;
```

Seguidamente, se enlazan estas variables con sus respectivos elementos en su archivo activity_main.xml:

```
mismascotasboton = (ImageButton) findViewById(R.id.idmismascotasbutton);
informacionboton = (ImageButton) findViewById(R.id.idinformacionbutton);
contactarboton = (ImageButton) findViewById(R.id.idcontactarbutton);
```

Tras esto, comenzamos a desarrollar las acciones de cada uno de los elementos.

Al pulsar el ImageButton “mismascotasboton”, se lanza un Intent (al que llamo gotopets) que abre la actividad “Pets Activity”:

```
    mismascotasboton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent gotopets = new Intent(MainActivity.this, PetsActivity.class);
            startActivity(gotopets);
        }
    });
```

Al pulsar el ImageButton “informacionboton”, se lanza un Intent (al que llamo gotoinf) que abre la actividad “Int Activity”:

```
informacionboton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent gotoinf = new Intent(MainActivity.this, InfActivity.class);
            startActivity(gotoinf);
        }
    });
```

Finalmente, al pulsar el ImageButton “contactarboton”, se lanza un Intent (al que llamo gotocontact) que abre la actividad “Contact Activity”:

```
contactarboton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent gotocontact = new Intent(MainActivity.this,
ContactActivity.class);
            startActivity(gotocontact);
        }
    });
```

2.5.2 DBHELPER

El archivo DBHelper.java no se complementa con ningún archivo xml, sino que crea la tabla, desde una extensión SQLiteOpenHelper, para crear la base de datos donde se almacenarán las mascotas que vayamos creando y sus respectivos datos.

El archivo DBHelper.java es el siguiente:

```
package com.example.doggycontrol;


import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteStatement;



public class DBHelper extends SQLiteOpenHelper {
    //Con esta String se crea la tabla
    String sql = "CREATE TABLE mascotas (id INTEGER PRIMARY KEY
AUTOINCREMENT, nombre1 TEXT,fecha1 TEXT, sexo1 TEXT, peso1 TEXT, chip1
TEXT, raza1 TEXT, pedigree1 TEXT, propiet1 TEXT, alergias1 TEXT,
vacunas1 TEXT, tratamientos1 TEXT, despint1 TEXT, despext1 TEXT,
masinf1 TEXT)";


    public DBHelper(Context context, String name,
SQLiteDatabase.CursorFactory factory, int version) {
        super(context, name, factory, version);
        //TODO Auto-generated constructor stub
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        //TODO Auto-generated constructor stub
        db.execSQL(sql);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
        //TODO Auto-generated constructor stub

    }

    //Creamos una consulta preparada para meter todos los datos en un
mismo grupo, ya que son muchos
    public void InsertarMascota(SQLiteDatabase db, String nombre1,
String fecha1, String sexo1, String peso1,String chip1, String raza1,
String pedigree1, String propiet1, String alergias1,String vacunas1,
String tratamientos1, String despint1, String despext1, String
masinf1) {
```

```
        SQLiteStatement pst = db.compileStatement("INSERT INTO
mascotas (nombre1, fecha1, sexo1, peso1, chip1, pedigree1, propiet1,
alergias1, vacunas1, tratamientos1, despint1, despext1, masinf1)
VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?)");
        //A cada interrogante le asignamos un valor
        pst.bindString(1, nombre1); //el primer interrogante equivale
a nombre1, y así sucesivamente
        pst.bindString(2, fecha1);
        pst.bindString(3, sexo1);
        pst.bindString(4, peso1);
        pst.bindString(5, chip1);
        pst.bindString(6, raza1);
        pst.bindString(7, pedigree1);
        pst.bindString(8, propiet1);
        pst.bindString(9, alergias1);
        pst.bindString(10, vacunas1);
        pst.bindString(11, tratamientos1);
        pst.bindString(12, despint1);
        pst.bindString(13, despext1);
        pst.bindString(14, masinf1);
        pst.execute();
    }
}
```

Lo primero que hacemos es crear la String sql para crear la tabla donde se almacenarán todos los datos:

```
String sql = "CREATE TABLE mascotas (id INTEGER PRIMARY KEY
AUTOINCREMENT, nombre1 TEXT,fecha1 TEXT, sexo1 TEXT, peso1 TEXT, chip1
TEXT, raza1 TEXT, pedigree1 TEXT, propiet1 TEXT, alergias1 TEXT, vacunas1
TEXT, tratamientos1 TEXT, despint1 TEXT, despext1 TEXT, masinf1 TEXT)";
```

Seguidamente, dentro del método onCreate agregamos una línea, la cual se encarga de que cuando se ponga en marcha la base de datos, si no hay una base de datos creada, la cree (la String donde se crea la tabla, recordemos, es sql):

```
@Override
public void onCreate(SQLiteDatabase db) {
    //TODO Auto-generated constructor stub
    db.execSQL(sql);
}
```

Como en DoggyControl se trabaja con muchos datos al mismo tiempo, he creado la consulta preparada “InsertarMascota”, la cual se encargará de trabajar con todos los datos como si fueran un “pack” y conseguir una programación más eficiente. Para ello, añadimos todas las String que se almacenarán en la tabla en un SQLiteStatement y, mediante bindString, iremos manejando los datos asignando a cada variable agregada al statement un

valor, que equivaldrá al lugar que ocupa cada dato en la tabla:

```
public void InsertarMascota(SQLiteDatabase db, String nombre1, String fecha1,
String sexo1, String peso1, String chip1, String raza1, String pedigree1, String
propiet1, String alergias1, String vacunas1, String tratamientos1, String
despint1, String despext1, String masinf1) {

      SQLiteStatement pst = db.compileStatement("INSERT INTO mascotas
(nombre1, fecha1, sexo1, peso1, chip1, raza1, pedigree1, propiet1, alergias1,
vacunas1, tratamientos1, despint1, despext1, masinf1) VALUES
(?,?,?,?,?,?,?,?,?,?,?,?,?,?)");
      //A cada interrogante le asignamos un valor
      pst.bindString(1, nombre1); //el primer interrogante equivale a nombre1, y
así sucesivamente
      pst.bindString(2, fecha1);
      pst.bindString(3, sexo1);
      pst.bindString(4, peso1);
      pst.bindString(5, chip1);
      pst.bindString(6, raza1);
      pst.bindString(7, pedigree1);
      pst.bindString(8, propiet1);
      pst.bindString(9, alergias1);
      pst.bindString(10, vacunas1);
      pst.bindString(11, tratamientos1);
      pst.bindString(12, despint1);
      pst.bindString(13, despext1);
      pst.bindString(14, masinf1);
      pst.execute();
   }
```

De esta forma, cuando queramos insertar una mascota nueva, basta con llamar a esta consulta preparada (InsertarMascota) para que se agreguen todos los datos a la tabla de la base de datos.

2.5.3 ADDACTIVITY

El archivo AddActivity.java es el que complementa al archivo activity_add.xml.

El archivo MainActivity.java es el siguiente:

```
package com.example.doggycontrol;

import androidx.annotation.Nullable;
import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;

import android.app.DatePickerDialog;
import android.app.Dialog;
```

```
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.database.sqlite.SQLiteDatabase;
import android.graphics.Bitmap;
import android.net.Uri;
import android.os.Bundle;
import android.provider.MediaStore;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.DatePicker;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.Spinner;
import android.widget.Toast;

import java.text.SimpleDateFormat;
import java.util.Calendar;


public class AddActivity extends AppCompatActivity {

    //PARA LA FECHA
    private int year, month, day;
    EditText fecha;
    Calendar C = Calendar.getInstance(); //para seleccionar la fecha
deseada
    DatePickerDialog.OnDateSetListener onDateSetListener;
    //FIN PARA LA FECHA




    // REVISA SI EL DISPOSITIVO TIENE CAMARA
    //(debido a que en AndroidManifest.xlm el requerimiento de
necesidad de camara es false)
    private boolean checkCameraHardware(Context context) {
        if
(context.getPackageManager().hasSystemFeature(PackageManager.FEATURE_C
AMERA)){
            // this device has a camera
            return true;
        } else {
            // no camera on this device
            return false;
        }
    }
    //FIN REVISA SI EL DISPOSITIVO TIENE CAMARA



    //PARA PONER UNA IMAGEN
    //Request code gallery
    private static final int GALLERY_REQUEST = 9;
    //Request code for camera
    private static final int CAMERA_REQUEST = 11;
    //Widgets
    private ImageView imagen;
```

```
    private Context context;
    //FIN PARA PONER UNA IMAGEN


    //PARA SPINNER DE SEXO
    //Añado la variable spinner1
    Spinner spinner1;
    //FIN PARA SPINNER DE SEXO


    //PARA SPINNER DE PEDIGREE
    //Añado la variable spinner2
    Spinner spinner2;
    //FIN PARA SPINNER DE PEDIGREE



    //PARA INSERTAR LOS DATOS EN LA DB AL DARLE A GUARDAR
    EditText nombre, peso, chip, raza, propiet, alergias, vacunas,
tratamientos, despint, despext, masinf;
    ImageView imagen1;
    Button guardar;
    //FIN PARA INSERTAR LOS DATOS EN LA DB AL DARLE A GUARDAR









    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_add);


        //PARA LA FECHA
        fecha = (EditText) findViewById(R.id.iddate);
        fecha.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {

                month = C.get(Calendar.MONTH);//para que al abrir se
ponga en la fecha actual
                day = C.get(Calendar.DAY_OF_MONTH);
                year = C.get(Calendar.YEAR);

                DatePickerDialog datePickerDialog = new
DatePickerDialog(view.getContext(), onDateSetListener, year, month,
day);
                datePickerDialog.show();
            }
        });
        onDateSetListener = new DatePickerDialog.OnDateSetListener() {
            @Override
            public void onDateSet(DatePicker view, int year, int
month, int dayOfMonth) {
                month = month+1;//se suma 1 al mes pq los meses se
indexan desde 0, siendo enero 0, febrero1...etc.
                fecha.setText(dayOfMonth+"/"+month+"/"+year);
```

```
            }
        };

        //FIN PARA LA FECHA


        //PARA PONER UNA IMAGEN
        context = this;
        //Initialize ImageView widget
        imagen = findViewById(R.id.idimagen);
        //Set onclick listener on ImageView
        imagen.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                //Open dialog
                showImageOptionDialog();
            }
        });
        //FIN PARA PONER UNA IMAGEN



        //PARA SPINNER DE SEXO
        //Codigo para agregar el menu desplegable del genero del
animal -spinner-
        spinner1 = findViewById(R.id.idgender);
        String [] genders={ "Macho","Hembra"};
        ArrayAdapter<String> adapter1 = new ArrayAdapter<>(this,
android.R.layout.simple_spinner_dropdown_item, genders);
        spinner1.setAdapter(adapter1);
        //FIN PARA SPINNER DE SEXO




        //PARA SPINNER DE PEDIGREE
        //Codigo para agregar el menu desplegable del genero del
animal -spinner-
        spinner2 = findViewById(R.id.idpedigree);
        String [] pedigree={ "Sí","No"};
        ArrayAdapter<String> adapter2 = new ArrayAdapter<>(this,
android.R.layout.simple_spinner_dropdown_item, pedigree);
        spinner2.setAdapter(adapter2);
        //FIN PARA SPINNER DE PEDIGREE




        //PARA INSERTAR LOS DATOS EN LA DB AL DARLE A GUARDAR
        nombre = findViewById(R.id.idname);
        fecha = findViewById(R.id.iddate);
        //fecha de nacimiento: fecha (ya puesto arriba)
        //sexo: spinner1 (ya puesto arriba)
        peso = findViewById(R.id.idpeso);
        chip = findViewById(R.id.idchip);
        raza = findViewById(R.id.idraza);
        //pedigree: spinner2 (ya puesto arriba)
        propiet = findViewById(R.id.idpropiet);
        alergias = findViewById(R.id.idalergias);
        vacunas = findViewById(R.id.idvacunas);
        tratamientos = findViewById(R.id.idtratamientos);
```

```
        despint = findViewById(R.id.iddespint);
        despext = findViewById(R.id.iddespext);
        masinf = findViewById(R.id.idmasinf);
        guardar = findViewById(R.id.idguardar);


        guardar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                String nombre1 = nombre.getText().toString();
                String fecha1 = fecha.getText().toString();
                String sexo1 = spinner1.getSelectedItem().toString();
                String peso1 = peso.getText().toString();
                String chip1 = chip.getText().toString();
                String raza1 = raza.getText().toString();
                String pedigree1 =
spinner2.getSelectedItem().toString();
                String propiet1 = propiet.getText().toString();
                String alergias1 = alergias.getText().toString();
                String vacunas1 = vacunas.getText().toString();
                String tratamientos1 =
tratamientos.getText().toString();
                String despint1 = despint.getText().toString();
                String despext1 = despext.getText().toString();
                String masinf1 = masinf.getText().toString();



                DBHelper dbhelper = new
DBHelper(getApplicationContext(), "DBMascotas", null, 1);
                SQLiteDatabase db = dbhelper.getWritableDatabase();
                dbhelper.InsertarMascota(db, nombre1, fecha1, sexo1,
peso1, chip1, raza1, pedigree1, propiet1, alergias1, vacunas1,
tratamientos1, despint1, despext1, masinf1);


                Toast.makeText(AddActivity.this, "Mascota agregada
correctamente", Toast.LENGTH_SHORT).show();
                //Vuelvo a la lista para ver los cambios
                Intent gotolist = new Intent(getApplicationContext(),
PetsActivity.class);
                startActivity(gotolist);
            }
        });

        //FIN PARA INSERTAR LOS DATOS EN LA DB AL DARLE A GUARDAR
    }


    //PARA PONER UNA IMAGEN
    private void showImageOptionDialog(){
        final String[] options =
getResources().getStringArray(R.array.image_options);
        AlertDialog.Builder builder = new
AlertDialog.Builder(context);
        builder.setTitle(R.string.alert_dialog_title)
                .setItems(options, new
DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int
```

```
which) {

                        switch (which){
                            case 0:
                                getImageFromGallery();
                                break;
                            case 1:
                                capturePictureFromCamera();
                                break;
                        }
                    }
                });

        AlertDialog dialog = builder.create();
        dialog.show();
    }

    private void capturePictureFromCamera(){
        Intent cameraIntent = new Intent();
        cameraIntent.setAction(MediaStore.ACTION_IMAGE_CAPTURE);
        startActivityForResult(cameraIntent, CAMERA_REQUEST);
    }

    //Open phone gallery
    private void getImageFromGallery(){
        Intent intent = new Intent();
        intent.setAction(Intent.ACTION_GET_CONTENT);
        intent.setType("image/*");
        startActivityForResult(intent, GALLERY_REQUEST);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode,
@Nullable Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        //Check if the intent was to pick image, was successful and an
image was picked
        if(requestCode == GALLERY_REQUEST && resultCode == RESULT_OK
&& data != null){

            //Get selected image uri from phone gallery
            Uri selectedImage = data.getData();

            //Display selected photo in image view
            imagen.setImageURI(selectedImage);
        }
        //Handle camera request
        else if(requestCode == CAMERA_REQUEST && resultCode ==
RESULT_OK && data != null){

            //We need a bitmap variable to store the photo
            Bitmap bitmap = (Bitmap) data.getExtras().get("data");

            //Display taken picture in image view
            imagen.setImageBitmap(bitmap);
        }
    }
    //FIN PARA PONER UNA IMAGEN

}
```

Como hay muchos aspectos a comentar en esta actividad (si nos fijamos, a lo largo del código éstos están separados por comentarios para intentar obtener un código más claro), vamos a ir por partes:

1. Agregar la fecha de nacimiento

Al pulsar en el campo de texto que corresponde a la fecha de nacimiento de la mascota, abrimos un diálogo con un DatePicker para seleccionarla.

Para ello, primero nombramos las variables:

```
//PARA LA FECHA
   private int year, month, day;
   EditText fecha;
   Calendar C = Calendar.getInstance(); //para seleccionar la fecha deseada
   DatePickerDialog.OnDateSetListener onDateSetListener;
 //FIN PARA LA FECHA
```

Seguidamente, se enlaza la variable del EditText con su respectivo elemento en su archivo activity_add.xml:

```
//PARA LA FECHA
fecha = (EditText) findViewById(R.id.iddate);
```

Tras esto, pasamos a programar el click en este elemento.

Al hacer click en “fecha”, hacemos que se abra el diálogo de DatePicker en la fecha actual. Esto se consigue cogiendo (con “get”) del Calendar del dispositivo sus datos.

Seguidamente, damos valor a las 3 variables del calendario [year (año), month (mes) y day (día)] eligiendo en éste la fecha deseada.

Una vez las variables han cogido un valor, sumamos una unidad a la variable “month”, esto es debido a que los meses se indexan desde el valor 0 (enero sería 0, febrero sería 1, etc). De esta forma, eligiendo el mes de enero, el EditText mostraría en la variable “month” 0+1=1= mes 1.

Tras esto, se pasa a mostrar en el campo la fecha seleccionada mediante un setText.

```
 fecha.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {

            month = C.get(Calendar.MONTH);//para que al abrir se ponga en la
fecha actual
            day = C.get(Calendar.DAY_OF_MONTH);
            year = C.get(Calendar.YEAR);

            DatePickerDialog datePickerDialog = new
DatePickerDialog(view.getContext(), onDateSetListener, year, month, day);
```

```
            datePickerDialog.show();
        }
    });
    onDateSetListener = new DatePickerDialog.OnDateSetListener() {
        @Override
        public void onDateSet(DatePicker view, int year, int month, int
dayOfMonth) {
            month = month+1;//se suma 1 al mes pq los meses se indexan desde 0,
siendo enero 0, febrero1...etc.
            fecha.setText(dayOfMonth+"/"+month+"/"+year);
        }
    };
    //FIN PARA LA FECHA
```

2. Opción de añadir imagen/fotografía al pulsar el ImageView

Al pulsar en el ImageView de la mascota se abre un diálogo en el que podremos seleccionar si queremos elegir una foto de la galería o si queremos tomar una foto. Para ello, primero debemos comprobar si el dispositivo tiene cámara (esto se debe a que en el AndroidManifest.xml habíamos declarado que el tener cámara no era un requisito indispensable para el uso de la aplicación):

```
// REVISA SI EL DISPOSITIVO TIENE CAMARA
//(debido a que en AndroidManifest.xlm el requerimiento de necesidad de
camara es false)
private boolean checkCameraHardware(Context context) {
    if
(context.getPackageManager().hasSystemFeature(PackageManager.FEATURE_C
AMERA)){
        // this device has a camera
        return true;
    } else {
        // no camera on this device
        return false;
    }
}
//FIN REVISA SI EL DISPOSITIVO TIENE CAMARA
```

Seguidamente, nombramos las variables que se utilizarán en este método:

```
//PARA PONER UNA IMAGEN
//Request code gallery
private static final int GALLERY_REQUEST = 9;
//Request code for camera
private static final int CAMERA_REQUEST = 11;
//Widgets
private ImageView imagen;
private Context context;
//FIN PARA PONER UNA IMAGEN
```

Tras esto, enlazamos la variable con su elemento y creamos su respectivo click, de

manera que cuando se pulse sobre él, se abrirá un diálogo:

```
//PARA PONER UNA IMAGEN
        context = this;
        //Initialize ImageView widget
        imagen = findViewById(R.id.idimagen);
        //Set onclick listener on ImageView
        imagen.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                //Open dialog
                showImageOptionDialog();
            }
        });
        //FIN PARA PONER UNA IMAGEN
```

Ahora, vamos a configurar el diálogo que se muestra. Para ello, cogemos las opciones de imagen y diálogo de Android Studio y agregamos dos opciones:

-Cuando se hace click en la primera opción del diálogo (case 0, lo que en el diálogo se llama por defecto “Choose photo from gallery”) se abre la galería para poder seleccionar una imagen.

-Cuando se hace click en la segunda opción del diálogo (case 1, lo que en el diálogo se llama por defecto “Take picture with camera”), se lanza la cámara para tomar una foto:

```
//PARA PONER UNA IMAGEN
    private void showImageOptionDialog(){
        final String[] options =
getResources().getStringArray(R.array.image_options);
        AlertDialog.Builder builder = new AlertDialog.Builder(context);
        builder.setTitle(R.string.alert_dialog_title)
                .setItems(options, new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {

                        switch (which){
                            case 0:
                                getImageFromGallery();
                                break;
                            case 1:
                                capturePictureFromCamera();
                                break;
                        }
                    }
                });

        AlertDialog dialog = builder.create();
        dialog.show();
    }
```

Si la opción elegida es la de tomar la foto con la cámara, se llama al método

capturePictureFromCamera(), donde llama a la acción *ACTION_IMAGE_CAPTURE* para capturar la imagen y se inicia el Intent(cameraIntent) de llamar a la cámara, bajo el código de solicitud *CAMERA_REQUEST:*

```
private void capturePictureFromCamera(){
    Intent cameraIntent = new Intent();
    cameraIntent.setAction(MediaStore.ACTION_IMAGE_CAPTURE);
    startActivityForResult(cameraIntent, CAMERA_REQUEST);
}
```

Si la opción elegida es la de elegir la imagen de la galería, se llama al método *getImageFromGallery(),* donde llama a la acción *ACTION_GET_CONTENT* para abrir la galería y se inicia el Intent(intent) de llamar a la galería para abrirla, bajo el código de solicitud *GALLERY_REQUEST*:

```
//Open phone gallery
private void getImageFromGallery(){
    Intent intent = new Intent();
    intent.setAction(Intent.ACTION_GET_CONTENT);
    intent.setType("image/*");
    startActivityForResult(intent, GALLERY_REQUEST);
}
```

Finalmente, en el método *OnActivityResult* se muestran los resultados de cada una de las opciones.

-Si la opción seleccionada obtiene el código de solicitud *GALLERY_REQUEST*, el resultado es correcto y la imagen (data) se escoge correctamente, se adquiere la imagen seleccionada en formato Uri y se coloca, mediante el método .setImageUri, en el ImageView, el cual hemos nombrado como “imagen”:

```
@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable
Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    //Check if the intent was to pick image, was successful and an image was
picked
    if(requestCode == GALLERY_REQUEST && resultCode == RESULT_OK &&
data != null){
        //Get selected image uri from phone gallery
        Uri selectedImage = data.getData();
        //Display selected photo in image view
        imagen.setImageURI(selectedImage);
    }
```

-Si la opción seleccionada obtiene el código de solicitud CAMERA_REQUEST, el resultado es correcto y la fotografía (data) es tomada correctamente, se adquiere la imagen tomada en un Bitmap para almacenarla y se coloca, mediante el método .setImageBitmap, en el ImageView, el cual hemos nombrado como “imagen”:

```
        //Handle camera request
        else if(requestCode == CAMERA_REQUEST && resultCode == RESULT_OK
&& data != null){
            //We need a bitmap variable to store the photo
            Bitmap bitmap = (Bitmap) data.getExtras().get("data");
            //Display taken picture in image view
            imagen.setImageBitmap(bitmap);
        }
    }
    //FIN PARA PONER UNA IMAGEN
```

3. Spinner: sexo y pedigree

En la pantalla de AddActivity.java nos encontramos con 2 spinner: sexo (cuyas opciones son “macho” o “hembra”) y pedigree (cuyas opciones son “sí” o “no”).

Primero, añado sus respectivas variables:

```
//PARA SPINNER DE SEXO
  Spinner spinner1;
//FIN PARA SPINNER DE SEXO

//PARA SPINNER DE PEDIGREE
   Spinner spinner2;
//FIN PARA SPINNER DE PEDIGREE
```

Seguidamente, enlazo las variables con sus elementos en activiy_add.xml.

Creo un String con las opciones deseadas y su respectivo adaptador (adapter1 para el sexo y adapter2 para el pedigree).

Finalmente, agrego el adaptador creado a su respectiva varible con .setAdapter():

```
        //PARA SPINNER DE SEXO
        //Codigo para agregar el menu desplegable del genero del animal -spinner-
        spinner1 = findViewById(R.id.idgender);
        String [] genders={ "Macho","Hembra"};
        ArrayAdapter<String> adapter1 = new ArrayAdapter<>(this,
android.R.layout.simple_spinner_dropdown_item, genders);
        spinner1.setAdapter(adapter1);
        //FIN PARA SPINNER DE SEXO


        //PARA SPINNER DE PEDIGREE
        //Codigo para agregar el menu desplegable del genero del animal -spinner-
        spinner2 = findViewById(R.id.idpedigree);
        String [] pedigree={ "Sí","No"};
        ArrayAdapter<String> adapter2 = new ArrayAdapter<>(this,
android.R.layout.simple_spinner_dropdown_item, pedigree);
        spinner2.setAdapter(adapter2);
        //FIN PARA SPINNER DE PEDIGREE
```

4. Insertar los datos en la base de datos al darle a “guardar”

Primero, nombro las variables a utilizar:

```
//PARA INSERTAR LOS DATOS EN LA DB AL DARLE A GUARDAR
   EditText nombre, peso, chip, raza, propiet, alergias, vacunas, tratamientos,
despint, despext, masinf;
   ImageView imagen1;
   Button guardar;
   //FIN PARA INSERTAR LOS DATOS EN LA DB AL DARLE A GUARDAR
```

Seguidamente, enlazo las variables con sus respectivos elementos de la interfaz:

```
//PARA INSERTAR LOS DATOS EN LA DB AL DARLE A GUARDAR
        nombre = findViewById(R.id.idname);
        fecha = findViewById(R.id.iddate);
        //fecha de nacimiento: fecha (ya puesto arriba)
        //sexo: spinner1 (ya puesto arriba)
        peso = findViewById(R.id.idpeso);
        chip = findViewById(R.id.idchip);
        raza = findViewById(R.id.idraza);
        //pedigree: spinner2 (ya puesto arriba)
        propiet = findViewById(R.id.idpropiet);
        alergias = findViewById(R.id.idalergias);
        vacunas = findViewById(R.id.idvacunas);
        tratamientos = findViewById(R.id.idtratamientos);
        despint = findViewById(R.id.iddespint);
        despext = findViewById(R.id.iddespext);
        masinf = findViewById(R.id.idmasinf);
        guardar = findViewById(R.id.idguardar);
```

Tras esto, pasamos a programar el click en el botón de “guardar”.

Al realizar este click, cogemos el texto que el usuario haya escrito en los campos de la interfaz y lo convertimos a string. Para ello, usamos el método .getText().toString(). Para los spinner se usa el método .getSelectedItem().toString(), ya que no es texto lo que aparece en este elemento, sino un ítem seleccionado de la lista del spinner.

Estos datos convertidos a String se almacenan en las variables que componen la base de datos que, como hemos visto en DBHelper, son: nombre1, fecha1, sexo1…etc.

```
Guardar.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {

            String nombre1 = nombre.getText().toString();
            String fecha1 = fecha.getText().toString();
            String sexo1 = spinner1.getSelectedItem().toString();
            String peso1 = peso.getText().toString();
            String chip1 = chip.getText().toString();
            String raza1 = raza.getText().toString();
            String pedigree1 = spinner2.getSelectedItem().toString();
            String propiet1 = propiet.getText().toString();
            String alergias1 = alergias.getText().toString();
            String vacunas1 = vacunas.getText().toString();
```

```
String tratamientos1 = tratamientos.getText().toString();
String despint1 = despint.getText().toString();
String despext1 = despext.getText().toString();
String masinf1 = masinf.getText().toString();
```

Seguidamente, creamos una clase de DBHelper a la que llamamos dbhelper. En ella, agregamos un nombre a la base de datos (DBMascotas):

```
DBHelper dbhelper = new DBHelper(getApplicationContext(), "DBMascotas",
null, 1);
```

Creamos un SQLiteDataBase que llamamos db, y accedemos a la base de datos dbhelper en modo escritura (getWritableDatabase)

```
SQLiteDatabase db = dbhelper.getWritableDatabase();
```

Se llama a la consulta preparada que hemos creado en el archivo DBHelper.java, InsertarMascota, con todas las variables que se ha de insertar en la base de datos. Tras realizar todo lo estipulado en la consulta preparada, se agrega InsertarMascota a nuestro DBHelper que hemos creado anteriormente (dbhelper). Añadiéndose así todos los datos que el usuario ha agregado a la base de datos de SQLite:

```
dbhelper.InsertarMascota(db, nombre1, fecha1, sexo1, peso1, chip1, raza1,
pedigree1, propiet1, alergias1, vacunas1, tratamientos1, despint1, despext1,
masinf1);
```

Finalmente, se lanza un *Toast* (mensaje que se muestra en pantalla durante unos segundos al usuario para luego volver a desaparecer automáticamente) y se llama a un Intent (gotolist) para que se nos lleve automáticamente a la actividad PetsActiviy.java y poder ver a la mascota agregada en la lista del ListView:

```
  Toast.makeText(AddActivity.this, "Mascota agregada correctamente",
Toast.LENGTH_SHORT).show();
            //Vuelvo a la lista para ver los cambios
            Intent gotolist = new Intent(getApplicationContext(), PetsActivity.class);
            startActivity(gotolist);
        }
    });
    //FIN PARA INSERTAR LOS DATOS EN LA DB AL DARLE A GUARDAR
```

2.5.4 BEANMASCOTAS

El archivo BeanMascotas.java no se complementa con ningún archivo xml, sino que ayuda a mostrar en el ListView los datos de las mascotas que han sido almacenados en la base de datos.

Este Bean será almacenado más tarde en un ArrayList para que sea mostrado en el ListView.

El archivo BeanMascotas.java es el siguiente:

```
package com.example.doggycontrol;

//recurrimos a un bean porque almacenaré este bean en un arraylist, de
esta forma se almacenarán todos los datos que vaya
//recogiendo en la base de datos

public class BeanMascotas {

    private int id; //lo agrego para poder eliminar, ya que elimino
sobre el id

    private String nombre1, fecha1, sexo1, peso1, chip1, raza1,
pedigree1, propiet1, alergias1, vacunas1, tratamientos1, despint1,
despext1, masinf1;

    public String getNombre() {
        return nombre1;
    }
    public void setNombre (String nombre1) {
        this.nombre1 = nombre1;
    }


    public String getFecha() { return fecha1; }
    public void setFecha (String fecha1) {this.fecha1 = fecha1;}

    public String getSexo() {return sexo1;}
    public void setSexo (String sexo1) {this.sexo1 = sexo1;}

    public String getPeso() {
        return peso1;
    }
    public void setPeso (String peso1) {
        this.peso1 = peso1;
    }

    public String getChip() {
        return chip1;
    }
    public void setChip (String chip1) {
        this.chip1 = chip1;
    }
```

```
    public String getRaza() {
        return raza1;
    }
    public void setRaza (String raza1) {
        this.raza1 = raza1;
    }

    public String getPedigree() {
        return pedigree1;
    }
    public void setPedigree (String pedigree1) {
        this.pedigree1 = pedigree1;
    }

    public String getPropiet() {
        return propiet1;
    }
    public void setPropiet (String propiet1) {
        this.propiet1 = propiet1;
    }

    public String getAlergias() {
        return alergias1;
    }
    public void setAlergias (String alergias1) {
        this.alergias1 = alergias1;
    }

    public String getVacunas() {
        return vacunas1;
    }
    public void setVacunas (String vacunas1) {
        this.vacunas1 = vacunas1;
    }

    public String getTratamientos() {
        return tratamientos1;
    }
    public void setTratamientos (String tratamientos1) {
        this.tratamientos1 = tratamientos1;
    }

    public String getDespint() {
        return despint1;
    }
    public void setDespint (String despint1) {
        this.despint1 = despint1;
    }

    public String getDespext() {
        return despext1;
    }
    public void setDespext (String despext1) {
        this.despext1 = despext1;
    }

    public String getMasinf() {
        return masinf1;
    }
    public void setMasinf (String masinf1) {
        this.masinf1 = masinf1;
```

```
    }



    public BeanMascotas (int id, String nombre1, String fecha1, String
sexo1, String peso1, String chip1, String raza1,String pedigree1,
String propiet1, String alergias1, String vacunas1, String
tratamientos1, String despint1, String despext1, String masinf1) {

        super();
        this.id = id;
        this.nombre1 = nombre1;
        this.fecha1 = fecha1;
        this.sexo1 = sexo1;
        this.peso1 = peso1;
        this.chip1 = chip1;
        this.raza1 = raza1;
        this.pedigree1 = pedigree1;
        this.propiet1 = propiet1;
        this.alergias1 = alergias1;
        this.vacunas1 = vacunas1;
        this.tratamientos1 = tratamientos1;
        this.despint1 = despint1;
        this.despext1 = despext1;
        this.masinf1 = masinf1;

    }



    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }



    //Se sobreescribe para que muestre el nombre del bean (de la
mascota en este caso) y no otro texto de codigo de aplicacion
    @Override
    public String toString() {
        return this.nombre1;
    }


}
```

Agrego como variable privada el id de cada mascota que, aunque no se agregue como valor a la tabla de la base de datos, será necesaria para poder eliminar los datos de la mascota que queramos, puesto que eliminaremos éstos en base al id:

```
private int id;
```

Seguidamente, agrego como String todas las variables que tendrá cada mascota y que serán las que se incluyan en la base de datos:

```
private String nombre1, fecha1, sexo1, peso1, chip1, raza1, pedigree1, propiet1,
alergias1, vacunas1, tratamientos1, despint1, despext1, masinf1;
```

En las líneas que siguen a estas variables, se muestran sus funciones, creando sus funciones “get” y “set”. Muestro aquí un ejemplo de la primera varible, String nombre1:

```
public String getNombre() {
    return nombre1;
}
public void setNombre (String nombre1) {
    this.nombre1 = nombre1;
}
```

Finalmente, se crea el constructor de BeanMascotas, en el cual se agrega en el campo deseado lo que el usuario haya añadido:

```
public BeanMascotas (int id, String nombre1, String fecha1, String sexo1, String
peso1, String chip1, String raza1,String pedigree1, String propiet1, String
alergias1, String vacunas1, String tratamientos1, String despint1, String
despext1, String masinf1) {
    super();
    this.id = id;
    this.nombre1 = nombre1;
    this.fecha1 = fecha1;
    this.sexo1 = sexo1;
    this.peso1 = peso1;
    this.chip1 = chip1;
    this.raza1 = raza1;
    this.pedigree1 = pedigree1;
    this.propiet1 = propiet1;
    this.alergias1 = alergias1;
    this.vacunas1 = vacunas1;
    this.tratamientos1 = tratamientos1;
    this.despint1 = despint1;
    this.despext1 = despext1;
    this.masinf1 = masinf1;
}
```

Finalmente, añadimos el método toString para que, cuando se muestre el nombre de la mascota en el ListView, se muestre el nombre en sí y no un código de la aplicación:

```
@Override
public String toString() {
    return this.nombre1;
}
```

2.5.5 PETSACTIVITY

El archivo PetsActivity.java es el que complementa al archivo activity_pets.xml.

El archivo PetsActivity.java es el siguiente:

```
package com.example.doggycontrol;


import androidx.annotation.RequiresApi;
import androidx.appcompat.app.AppCompatActivity;


import android.content.Context;
import android.content.Intent;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Build;
import android.os.Bundle;
import android.os.VibrationEffect;
import android.os.Vibrator;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ImageButton;
import android.widget.ListView;
import android.widget.Toast;

import java.util.ArrayList;


public class PetsActivity extends AppCompatActivity {

    ImageButton agregarmascotaboton;


    ListView lv;
    ArrayList<BeanMascotas> lista = new ArrayList<BeanMascotas>();
    BeanMascotas mascota;
    SQLiteDatabase db;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_pets);

        //IR A LA PAGINA DE AGREGAR MASCOTA CUANDO SE PULSE EL
IMAGEBUTTON
        agregarmascotaboton = (ImageButton)
findViewById(R.id.idagregarmascotabutton);

        agregarmascotaboton.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent gotoadd = new Intent(PetsActivity.this,
```

```
AddActivity.class);

                startActivity(gotoadd);
            }
        });
        //FIN IR A LA PAGINA DE AGREGAR MASCOTA CUANDO SE PULSE EL
IMAGEBUTTON



        //PARA QUE LOS DATOS DE LA DB SE PONGAN EN EL LISTVIEW
        lv = findViewById(R.id.lista_mascotas);

        DBHelper dbhelper = new DBHelper(getApplicationContext(),
"DBMascotas", null, 1);
        //accedemos a la database en modo lectura y tiramos un cursor
que lea y recupere los datos
        db = dbhelper.getWritableDatabase();
        //el * significa t o d o  (para que lea to d o s los datos
        Cursor c = db.rawQuery("SELECT * FROM mascotas", null);

        if (c.moveToFirst()) {
            do {
                mascota = new BeanMascotas(c.getShort(0),
c.getString(1), c.getString(2), c.getString(3),c.getString(4),
c.getString(5), c.getString(6), c.getString(7),c.getString(8),
c.getString(9), c.getString(10), c.getString(11),c.getString(12),
c.getString(13), c.getString(14));
                lista.add(mascota);
            } while (c.moveToNext()); //mientras haya un siguiente
registro
        } else {
            Toast.makeText(PetsActivity.this, "No hay registros en la
base de datos", Toast.LENGTH_SHORT).show();
        }

        ArrayAdapter<BeanMascotas> adap = new
ArrayAdapter<BeanMascotas>(getApplicationContext(),
android.R.layout.simple_list_item_1, lista);
        lv.setAdapter(adap);
        //FIN PARA QUE LOS DATOS DE LA DB SE PONGAN EN EL LISTVIEW




        //ELIMINAR DE DB--METODO PARA CUANDO MANTENGAMOS PULSADO UN
ITEM DEL LA LISTA MAS DE 1 SEGUNDO (ELIMINARLO)
        lv.setOnItemLongClickListener(new
AdapterView.OnItemLongClickListener() {
            @RequiresApi(api = Build.VERSION_CODES.O) //requerimiento
de api para vibrar
            @Override //nos interesa el int arg2 que es el que tiene
la posición
            public boolean onItemLongClick(AdapterView<?> arg0, View
arg1, int arg2, long arg3) {

                //para que el telefono vibre
                Vibrator v = (Vibrator)
getSystemService(Context.VIBRATOR_SERVICE);
                //Vibrar por 300milisegundos --> 0.3segundos
```

```
                v.vibrate(VibrationEffect.createOneShot(300,
VibrationEffect.DEFAULT_AMPLITUDE));
                //fin para que el telefono vibre


                //Declaramos el String con lo que vamos a borrar, en
este caso t.o.d.o con la coincidencia del id
                String sql = "DELETE FROM mascotas WHERE
id="+lista.get(arg2).getId();
                db.execSQL(sql); //lo ejecutamos
                Toast.makeText(PetsActivity.this, "Mascota eliminada
correctamente", Toast.LENGTH_SHORT).show();
                Intent i = new Intent (PetsActivity.this,
PetsActivity.class);
                startActivity(i); //con este intent llamamos a la
misma actividad, para que se recargue la actividad en la que estoy
                finish();//Terminamos esta actvidad
                return false;
            }
        });
        //FIN ELIMINAR DE DB--METODO PARA CUANDO MANTENGAMOS PULSADO
UN ITEM DEL LA LISTA MAS DE 1 SEGUNDO (ELIMINARLO)




        //ACTUALIZAR DE DB--METODO PARA CUANDO PULSEMOS UN ITEM DEL LA
LISTA (EDITARLO O VER SU INFORMACION)
        lv.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> arg0, View arg1,
int arg2, long arg3) {
                //Recojo los datos que quiero pasar a la otra
actividad
                int id = lista.get(arg2).getId();
                String nombre1 = lista.get(arg2).getNombre();
                String fecha1 = lista.get(arg2).getFecha();
                String sexo1 = lista.get(arg2).getSexo();
                String peso1 = lista.get(arg2).getPeso();
                String chip1 = lista.get(arg2).getChip();
                String raza1 = lista.get(arg2).getRaza();
                String pedigree1 = lista.get(arg2).getPedigree();
                String propiet1 = lista.get(arg2).getPropiet();
                String alergias1 = lista.get(arg2).getAlergias();
                String vacunas1 = lista.get(arg2).getVacunas();
                String tratamientos1 =
lista.get(arg2).getTratamientos();
                String despint1 = lista.get(arg2).getDespint();
                String despext1 = lista.get(arg2).getDespext();
                String masinf1 = lista.get(arg2).getMasinf();

                //Creo el intent y seteo esos datos
                Intent in = new Intent(getApplicationContext(),
DetailsActivity.class);
                in.putExtra("id", id);
                in.putExtra("nombre1", nombre1);
                in.putExtra("fecha1", fecha1);
                in.putExtra("sexo1", sexo1);
                in.putExtra("peso1", peso1);
```

```
                in.putExtra("chip1", chip1);
                in.putExtra("raza1", raza1);
                in.putExtra("pedigree1", pedigree1);
                in.putExtra("propiet1", propiet1);
                in.putExtra("alergias1", alergias1);
                in.putExtra("vacunas1", vacunas1);
                in.putExtra("tratamientos1", tratamientos1);
                in.putExtra("despint1", despint1);
                in.putExtra("despext1", despext1);
                in.putExtra("masinf1", masinf1);

                //Lanzo la actividad
                startActivity(in);
            }
        });
        //FIN ACTUALIZAR DE DB--METODO PARA CUANDO PULSEMOS UN ITEM
DEL LA LISTA (EDITARLO O VER SU INFORMACION)
    }
}
```

Tal y como ocurrió en el archivo AddActivity.java, hay muchos aspectos a comentar en esta actividad (si nos fijamos, a lo largo del código éstos están separados por comentarios para intentar obtener un código más claro), vamos a ir por partes:

1. Ir a la actividad AddActivity.java cuando se pulse su respectivo ImageButton

Para ello, nombramos la variable y la enlazamos con su respectivo elemento en el layout. Finalmente, programamos el click en este ImageButton y lanzamos el Intent(gotoadd) para que nos lleve a la actividad AddActivity.java:

```
ImageButton agregarmascotaboton;
//IR A LA PAGINA DE AGREGAR MASCOTA CUANDO SE PULSE EL
IMAGEBUTTON
agregarmascotaboton = (ImageButton)
findViewById(R.id.idagregarmascotabutton);

    agregarmascotaboton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent gotoadd = new Intent(PetsActivity.this, AddActivity.class);
            startActivity(gotoadd);
        }
    });
    //FIN IR A LA PAGINA DE AGREGAR MASCOTA CUANDO SE PULSE EL
IMAGEBUTTON
```

2. Hacer que los datos que se encuentran en la base de datos aparezcan en el ListView

Para ello, comenzamos nombrando las variables y métodos que vamos a usar. Nombramos el ListView, la base de datos SQLiteDatabase y el BeanMascota. Además, agregamos el archivo BeanMascota.java a un ArrayList para su uso, tal y como lo comenté en su archivo:

```
ListView lv;
ArrayList<BeanMascotas> lista = new ArrayList<BeanMascotas>();
BeanMascotas mascota;
SQLiteDatabase db;
```

Enlazamos la variable lv a su respectivo elemento:

```
//PARA QUE LOS DATOS DE LA DB SE PONGAN EN EL LISTVIEW
      lv = findViewById(R.id.lista_mascotas);
```

Seguidamente, creamos la clase dbhelper y accedemos a ella en modo escritura, igual que lo habíamos hecho en la actividad AddActivity.java. Además, tiramos un Cursor que se encargará de leer y recuperar todos los datos que se encuentran en la base de datos:

```
DBHelper dbhelper = new DBHelper(getApplicationContext(), "DBMascotas",
null, 1);
      //accedemos a la database en modo lectura y tiramos un cursor que lea y
recupere los datos
      db = dbhelper.getWritableDatabase();
      //el * significa t o d o  (para que lea todos los datos
      Cursor c = db.rawQuery("SELECT * FROM mascotas", null);
```

El Cursor se coloca al inicio de la base de datos (c.moveToFirst) y, mientras que se pueda mover al siguiente registro (c.moveToNext, -lo que significa que no ha llegado al final de los elementos de la base de datos-), se crea una nueva mascota mediante el bean BeanMascotas.java ya explicado, adquiriendo (mediante c.getString(*índice de la variable*) y llevando finalmente todos los datos de la base de datos al ListView.

```
if (c.moveToFirst()) {
          do {
             mascota = new BeanMascotas(c.getShort(0), c.getString(1),
c.getString(2), c.getString(3),c.getString(4), c.getString(5), c.getString(6),
c.getString(7),c.getString(8), c.getString(9), c.getString(10),
c.getString(11),c.getString(12), c.getString(13), c.getString(14));
             lista.add(mascota);
          } while (c.moveToNext()); //mientras haya un siguiente registro
```

Cuando no haya registros en la base de datos, se muestra un Toast, informándolo:

```
} else {
        Toast.makeText(PetsActivity.this, "No hay registros en la base de datos",
Toast.LENGTH_SHORT).show();
    }
```

Finalmente, se agrega esta información a un adaptador simple (adap) de ListView y se agrega dicho adaptador a ListView (lv):

```
ArrayAdapter<BeanMascotas> adap = new
ArrayAdapter<BeanMascotas>(getApplicationContext(),
android.R.layout.simple_list_item_1, lista);
    lv.setAdapter(adap);
    //FIN PARA QUE LOS DATOS DE LA DB SE PONGAN EN EL LISTVIEW
```

3. Mantener pulsado una mascota para eliminarla y que el dispositivo vibre

Al eliminar una mascota de la base de datos, la estamos eliminando también del ListView.

Primero, creamos un click largo de un ítem del ListView (setOnItemLongClickListener) y agregamos el requerimiento de API para poder vibrar:

```
//ELIMINAR DE DB--METODO PARA CUANDO MANTENGAMOS PULSADO UN
ITEM DEL LA LISTA MAS DE 1 SEGUNDO (ELIMINARLO)
    lv.setOnItemLongClickListener(new
AdapterView.OnItemLongClickListener() {
        @RequiresApi(api = Build.VERSION_CODES.O) //requerimiento de api
para vibrar
```

Llamamos una clase booleana para llevar a cabo el click largo y, de éste, nos interesa su variable arg2, que es la variable que se encarga de la posición. Dentro de esta clase, llamamos al vibrador del sistema (v) y le aportamos la vibración de amplitud por defecto y que vibre durante 0.3 segundos:

```
@Override //nos interesa el int arg2 que es el que tiene la posición
        public boolean onItemLongClick(AdapterView<?> arg0, View arg1, int
arg2, long arg3) {
            //para que el telefono vibre
            Vibrator v = (Vibrator)
getSystemService(Context.VIBRATOR_SERVICE);
            //Vibrar por 300milisegundos --> 0.3segundos
            v.vibrate(VibrationEffect.createOneShot(300,
VibrationEffect.DEFAULT_AMPLITUDE));
            //fin para que el telefono vibre
```

Ya hemos conseguido que el dispositivo vibre al mantener pulsado un elemento del ListView, ahora vamos a ver como conseguir que se eliminen sus respectivos datos.

Dentro de la clase booleana que hemos creado anteriormente, declaramos la String sql mediante la cual vamos a proceder a eliminar, eliminando todos los datos que concuerden con el id de la mascota que mantengamos pulsada, y lo ejecutamos:

```
//Declaramos el String con lo que vamos a borrar, en este caso todo con la coincidencia del id
            String sql = "DELETE FROM mascotas WHERE id="+lista.get(arg2).getId();
db.execSQL(sql);
```

Finalmente, lanzamos un *Toast* que se mostrará al usuario confirmando que la mascota ha sido eliminada y se lanzará un Intent(i) que recargará la actividad PetsActivity.java para que se pueda visualizar la lista sin la mascota que hemos eliminado:

```
Toast.makeText(PetsActivity.this, "Mascota eliminada correctamente", Toast.LENGTH_SHORT).show();
            Intent i = new Intent (PetsActivity.this, PetsActivity.class);
            startActivity(i);
            finish();//Terminamos esta actividad
            return false;
        }
    });
    //FIN ELIMINAR DE DB--METODO PARA CUANDO MANTENGAMOS PULSADO UN ITEM DEL LA LISTA MAS DE 1 SEGUNDO (ELIMINARLO)
```

4. Al pulsar un elemento de la lista llevar los datos y abrir la actividad DetailsActivity.java

Cuando pulsemos en un elemento de la lista llamamos al método onItemClick y recogemos todos los datos (esto se realiza mediante su posición y su función get que hemos declarado en el archivo BeanMascotas.java) para poder pasarlos a la siguiente actividad:

```
//ACTUALIZAR DE DB--METODO PARA CUANDO PULSEMOS UN ITEM DEL LA LISTA (EDITARLO O VER SU INFORMACION)
    lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> arg0, View arg1, int arg2, long
```

```
arg3) {
            //Recojo los datos que quiero pasar a la otra actividad
            int id = lista.get(arg2).getId();
            String nombre1 = lista.get(arg2).getNombre();
            String fecha1 = lista.get(arg2).getFecha();
            String sexo1 = lista.get(arg2).getSexo();
            String peso1 = lista.get(arg2).getPeso();
            String chip1 = lista.get(arg2).getChip();
            String raza1 = lista.get(arg2).getRaza();
            String pedigree1 = lista.get(arg2).getPedigree();
            String propiet1 = lista.get(arg2).getPropiet();
            String alergias1 = lista.get(arg2).getAlergias();
            String vacunas1 = lista.get(arg2).getVacunas();
            String tratamientos1 = lista.get(arg2).getTratamientos();
            String despint1 = lista.get(arg2).getDespint();
            String despext1 = lista.get(arg2).getDespext();
            String masinf1 = lista.get(arg2).getMasinf();
```

Finalmente, creo el Intent(in) que se encargará de colocar, mediante putExtra, los datos recogidos en la actividad DetailsActivity.java y se nos abrirá esa actividad. Ya tendríamos abierta la actividad DetailsActivity.java con todos los datos situados en su respectivo campo:

```
//Creo el intent y seteo esos datos
            Intent in = new Intent(getApplicationContext(), DetailsActivity.class);
            in.putExtra("id", id);
            in.putExtra("nombre1", nombre1);
            in.putExtra("fecha1", fecha1);
            in.putExtra("sexo1", sexo1);
            in.putExtra("peso1", peso1);
            in.putExtra("chip1", chip1);
            in.putExtra("raza1", raza1);
            in.putExtra("pedigree1", pedigree1);
            in.putExtra("propiet1", propiet1);
            in.putExtra("alergias1", alergias1);
            in.putExtra("vacunas1", vacunas1);
            in.putExtra("tratamientos1", tratamientos1);
            in.putExtra("despint1", despint1);
            in.putExtra("despext1", despext1);
            in.putExtra("masinf1", masinf1);
            //Lanzo la actividad
            startActivity(in);
        }
    });
    //FIN ACTUALIZAR DE DB--METODO PARA CUANDO PULSEMOS UN ITEM
DEL LA LISTA (EDITARLO O VER SU INFORMACION)
```

2.5.6 DETAILSACTIVITY

El archivo DetailsActivity.java es el que complementa al archivo activity_details.xml.

No comentaré muchos aspectos de este archivo ya que son similares a la actividad AddActivity.java (seleccionar fecha, seleccionar imagen/tomar fotografía, spinner…).
Al igual que en su archivo complementario xml, se propondrá, como trabajo futuro, crear un único archivo java para ambas funciones (añadir mascota y editar/actualizar mascota). Ya que, de esta manera, los cambios futuros se podrían realizar en el mismo archivo en el que se agregan los datos iniciales de las mascotas.

El único aspecto a comentar de este apartado es el actualizar la información de la mascota.

El archivo DetailsActivity.java es el siguiente:

```
package com.example.doggycontrol;

import android.app.DatePickerDialog;
import android.app.Dialog;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.database.sqlite.SQLiteDatabase;
import android.graphics.Bitmap;
import android.net.Uri;
import android.os.Bundle;
import android.provider.MediaStore;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.DatePicker;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.Spinner;
import android.widget.Toast;

import androidx.annotation.Nullable;
import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;

import java.text.SimpleDateFormat;
import java.util.ArrayList;
```

```
import java.util.Calendar;

public class DetailsActivity extends AppCompatActivity {


    EditText ednombre, edpeso, edchip, edraza, edpropiet, edalergias,
edvacunas,edtratamientos, eddespint, eddespext, edmasinf;
    Spinner edspinner1, edspinner2;
    Button actualizar;
    String nombre2, fecha2, sexo2, peso2, chip2, raza2, pedigree2,
propiet2, alergias2, vacunas2,tratamientos2, despint2, despext2,
masinf2;
    int id;
    SQLiteDatabase db;



    //PARA LA FECHA
    private int year, month, day;
    EditText _fecha;
    Calendar C = Calendar.getInstance(); //para seleccionar la fecha
deseada
    DatePickerDialog.OnDateSetListener onDateSetListener;
    //FIN PARA LA FECHA




    // REVISA SI EL DISPOSITIVO TIENE CAMARA
    //(debido a que en AndroidManifest.xlm el requerimiento de
necesidad de camara es false)
    private boolean checkCameraHardware(Context context) {
        if
(context.getPackageManager().hasSystemFeature(PackageManager.FEATURE_C
AMERA)){
            // this device has a camera
            return true;
        } else {
            // no camera on this device
            return false;
        }
    }
    //FIN REVISA SI EL DISPOSITIVO TIENE CAMARA




    //PARA PONER UNA IMAGEN
    //Request code gallery
    private static final int GALLERY_REQUEST = 9;
    //Request code for camera
    private static final int CAMERA_REQUEST = 11;
    //Widgets
    private ImageView _imagen;
    private Context context;
    //FIN PARA PONER UNA IMAGEN


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_details);



        //PARA LA FECHA
        _fecha = (EditText) findViewById(R.id.editfecha);
        _fecha.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {

                month = C.get(Calendar.MONTH);//para que al abrir se
ponga en la fecha actual
                day = C.get(Calendar.DAY_OF_MONTH);
                year = C.get(Calendar.YEAR);

                DatePickerDialog datePickerDialog = new
DatePickerDialog(view.getContext(), onDateSetListener, year, month,
day);
                datePickerDialog.show();
            }
        });
        onDateSetListener = new DatePickerDialog.OnDateSetListener() {
            @Override
            public void onDateSet(DatePicker view, int year, int
month, int dayOfMonth) {
                month = month+1;//se suma 1 al mes pq los meses se
indexan desde 0, siendo enero 0, febrero1...etc.
                //si no sumo 1 y selecciono agosto, sale el mes 7 en
vez del 8 en el edittext.
                _fecha.setText(dayOfMonth+"/"+month+"/"+year);
            }
        };

        //FIN PARA LA FECHA




        //PARA PONER UNA IMAGEN
        context = this;
        //Initialize ImageView widget
        _imagen = findViewById(R.id.editimagen);
        //Set onclick listener on ImageView
        _imagen.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                //Open dialog
                showImageOptionDialog();
            }
        });
        //FIN PARA PONER UNA IMAGEN



        //PARA SPINNER DE SEXO
        //Codigo para agregar el menu desplegable del genero del
animal -spinner-
```

```
        edspinner1 = findViewById(R.id.editsexo);
        String [] genders={ "Macho","Hembra"};
        ArrayAdapter<String> _adapter1 = new ArrayAdapter<>(this,
android.R.layout.simple_spinner_dropdown_item, genders);
        edspinner1.setAdapter(_adapter1);

        //las siguientes lineas son para que se actualice el spinner
con la opcion elegida en la ventana de actualizar mascota
        ArrayList<String> arr1 = new ArrayList<String>(3);
        // añadir valores
        arr1.add("Macho");
        arr1.add("Hembra");
        //FIN PARA SPINNER DE SEXO




        //PARA SPINNER DE PEDIGREE
        //Codigo para agregar el menu desplegable del genero del
animal -spinner-
        edspinner2 = findViewById(R.id.editpedigree);
        String [] pedigree={ "Sí","No"};
        ArrayAdapter<String> _adapter2 = new ArrayAdapter<>(this,
android.R.layout.simple_spinner_dropdown_item, pedigree);
        edspinner2.setAdapter(_adapter2);

        //las siguientes lineas son para que se actualice el spinner
con la opcion elegida en la ventana de actualizar mascota
        ArrayList<String> arr2 = new ArrayList<String>(3);
        // añadir valores
        arr2.add("Sí");
        arr2.add("No");
        //FIN PARA SPINNER DE PEDIGREE




        //PARA ACTUALIZAR LA INFORMACION DE LA MASCOTA
        ednombre = findViewById(R.id.editnombre);
        _fecha = findViewById(R.id.editfecha);
        //fecha de nacimiento: _fecha (ya puesto arriba)
        //sexo: edspinner1 (ya puesto arriba)
        edpeso = findViewById(R.id.editpeso);
        edchip = findViewById(R.id.editchip);
        edraza = findViewById(R.id.editraza);
        //pedigree: edspinner2 (ya puesto arriba)
        edpropiet = findViewById(R.id.editpropiet);
        edalergias = findViewById(R.id.editalergias);
        edvacunas = findViewById(R.id.editvacunas);
        edtratamientos = findViewById(R.id.edittratamientos);
        eddespint = findViewById(R.id.editdespint);
        eddespext = findViewById(R.id.editdespext);
        edmasinf = findViewById(R.id.editmasinf);
        actualizar= findViewById(R.id.idactualizar);


        //Recojo los datos que he pasado en la otra actividad y se los
seteo a los EditText
        Bundle bun = getIntent().getExtras(); //esto es un objeto
general
        nombre2 = bun.getString("nombre1");
```

```
        fecha2 = bun.getString("fecha1");
        sexo2 = bun.getString("sexo1");
        peso2 = bun.getString("peso1");
        chip2 = bun.getString("chip1");
        raza2 = bun.getString("raza1");
        pedigree2 = bun.getString("pedigree1");
        propiet2 = bun.getString("propiet1");
        alergias2 = bun.getString("alergias1");
        vacunas2 = bun.getString("vacunas1");
        tratamientos2 = bun.getString("tratamientos1");
        despint2 = bun.getString("despint1");
        despext2 = bun.getString("despext1");
        masinf2 = bun.getString("masinf1");

        id = bun.getInt("id");

        ednombre.setText(nombre2);
        _fecha.setText(fecha2);
        edspinner1.setSelection(arr1.indexOf(sexo2));
        edpeso.setText(peso2);
        edchip.setText(chip2);
        edraza.setText(raza2);
        edspinner2.setSelection(arr2.indexOf(pedigree2));
        edpropiet.setText(propiet2);
        edalergias.setText(alergias2);
        edvacunas.setText(vacunas2);
        edtratamientos.setText(tratamientos2);
        eddespint.setText(despint2);
        eddespext.setText(despext2);
        edmasinf.setText(masinf2);

        //Al pulsar el boton actualizar
        actualizar.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View arg0) {
                //creo la conexion y acceso en modo escritura
                DBHelper dbhelper = new
DBHelper(getApplicationContext(), "DBMascotas", null, 1);
                db = dbhelper.getWritableDatabase();
                //declaro las variables que actualizaran el registro
en la base de datos con lo que hemos puesto en los edittext
                String nombre1 = ednombre.getText().toString();
                String fecha1 = _fecha.getText().toString();
                String sexo1 =
edspinner1.getSelectedItem().toString();
                String peso1 = edpeso.getText().toString();
                String chip1 = edchip.getText().toString();
                String raza1 = edraza.getText().toString();
                String pedigree1 =
edspinner2.getSelectedItem().toString();
                String propiet1 = edpropiet.getText().toString();
                String alergias1 = edalergias.getText().toString();
                String vacunas1 = edvacunas.getText().toString();
                String tratamientos1 =
edtratamientos.getText().toString();
                String despint1 = eddespint.getText().toString();
                String despext1 = eddespext.getText().toString();
                String masinf1 = edmasinf.getText().toString();
                //creo que String de actualización... (el statement no
sirve con el update, solo sirve para modificar
```

```
                // las tablas, no el contenido de las tablas)
                String sql = "UPDATE mascotas SET
nombre1='"+nombre1+"',fecha1='"+fecha1+"',sexo1='"+sexo1+"',peso1='"+p
eso1+"',chip1='"+chip1+"',raza1='"+raza1+"',pedigree1='"+pedigree1+"',
propiet1='"+propiet1+"',alergias1='"+alergias1+"',vacunas1='"+vacunas1
+"',tratamientos1='"+tratamientos1+"',despint1='"+despint1+"',despext1
='"+despext1+"',masinf1='"+masinf1+"' WHERE id="+id;
                //lo ejecuto

                if (db != null) {
                    db.execSQL(sql);
                    Toast.makeText(DetailsActivity.this, "Mascota
actualizada correctamente", Toast.LENGTH_SHORT).show();
                    //Vuelvo a la lista para ver los cambios
                    Intent in = new Intent(getApplicationContext(),
PetsActivity.class);
                    startActivity(in);
                } else {
                    Toast.makeText(DetailsActivity.this, "Error al
actualizar", Toast.LENGTH_SHORT).show();
                }


            }

        });
        //FIN PARA ACTUALIZAR LA INFORMACION DE LA MASCOTA

    }


    //PARA PONER UNA IMAGEN
    private void showImageOptionDialog(){
        final String[] options =
getResources().getStringArray(R.array.image_options);
        AlertDialog.Builder builder = new
AlertDialog.Builder(context);
        builder.setTitle(R.string.alert_dialog_title)
                .setItems(options, new
DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int
which) {

                        switch (which){
                            case 0:
                                getImageFromGallery();
                                break;
                            case 1:
                                capturePictureFromCamera();
                                break;
                        }
                    }
                });

        AlertDialog dialog = builder.create();
        dialog.show();
    }

    private void capturePictureFromCamera(){
        Intent cameraIntent = new Intent();
```

```
        cameraIntent.setAction(MediaStore.ACTION_IMAGE_CAPTURE);
        startActivityForResult(cameraIntent, CAMERA_REQUEST);
    }

    //Open phone gallery
    private void getImageFromGallery(){
        Intent intent = new Intent();
        intent.setAction(Intent.ACTION_GET_CONTENT);
        intent.setType("image/*");
        startActivityForResult(intent, GALLERY_REQUEST);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode,
@Nullable Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        //Check if the intent was to pick image, was successful and an
image was picked
        if(requestCode == GALLERY_REQUEST && resultCode == RESULT_OK
&& data != null){

            //Get selected image uri from phone gallery
            Uri selectedImage = data.getData();

            //Display selected photo in image view
            _imagen.setImageURI(selectedImage);
        }
        //Handle camera request
        else if(requestCode == CAMERA_REQUEST && resultCode ==
RESULT_OK && data != null){

            //We need a bitmap variable to store the photo
            Bitmap bitmap = (Bitmap) data.getExtras().get("data");

            //Display taken picture in image view
            _imagen.setImageBitmap(bitmap);
        }
    }
    //FIN PARA PONER UNA IMAGEN

}
```

Primero, nombro las variables que se usarán en esta actividad:

```
  EditText ednombre, edpeso, edchip, edraza, edpropiet, edalergias,
edvacunas,edtratamientos, eddespint, eddespext, edmasinf;
   Spinner edspinner1, edspinner2;
   Button actualizar;
   String nombre2, fecha2, sexo2, peso2, chip2, raza2, pedigree2, propiet2,
alergias2, vacunas2,tratamientos2, despint2, despext2, masinf2;
   int id;
   SQLiteDatabase db;
```

Seguidamente, enlazo las variables a sus respectivos elementos del layout:

```
//PARA ACTUALIZAR LA INFORMACION DE LA MASCOTA
      ednombre = findViewById(R.id.editnombre);
      _fecha = findViewById(R.id.editfecha);
```

```
//fecha de nacimiento: _fecha (ya puesto arriba)
//sexo: edspinner1 (ya puesto arriba)
edpeso = findViewById(R.id.editpeso);
edchip = findViewById(R.id.editchip);
edraza = findViewById(R.id.editraza);
//pedigree: edspinner2 (ya puesto arriba)
edpropiet = findViewById(R.id.editpropiet);
edalergias = findViewById(R.id.editalergias);
edvacunas = findViewById(R.id.editvacunas);
edtratamientos = findViewById(R.id.edittratamientos);
eddespint = findViewById(R.id.editdespint);
eddespext = findViewById(R.id.editdespext);
edmasinf = findViewById(R.id.editmasinf);
actualizar= findViewById(R.id.idactualizar);
```

Recojo los datos que he cogido y pasado desde la actividad PetsActivity.java hacia ésta mediante un objeto Bundle:

```
//Recojo los datos que he pasado en la otra actividad y se los seteo a los EditText
Bundle bun = getIntent().getExtras(); //esto es un objeto general
nombre2 = bun.getString("nombre1");
fecha2 = bun.getString("fecha1");
sexo2 = bun.getString("sexo1");
peso2 = bun.getString("peso1");
chip2 = bun.getString("chip1");
raza2 = bun.getString("raza1");
pedigree2 = bun.getString("pedigree1");
propiet2 = bun.getString("propiet1");
alergias2 = bun.getString("alergias1");
vacunas2 = bun.getString("vacunas1");
tratamientos2 = bun.getString("tratamientos1");
despint2 = bun.getString("despint1");
despext2 = bun.getString("despext1");
masinf2 = bun.getString("masinf1");

id = bun.getInt("id");
```

Agrego a los campos de texto (.setText()) y spinners (.setSelection()) los datos recogidos:

```
ednombre.setText(nombre2);
_fecha.setText(fecha2);
edspinner1.setSelection(arr1.indexOf(sexo2));
edpeso.setText(peso2);
edchip.setText(chip2);
edraza.setText(raza2);
edspinner2.setSelection(arr2.indexOf(pedigree2));
edpropiet.setText(propiet2);
edalergias.setText(alergias2);
edvacunas.setText(vacunas2);
edtratamientos.setText(tratamientos2);
eddespint.setText(despint2);
eddespext.setText(despext2);
```

```
edmasinf.setText(masinf2);
```

Al pulsar en el botón “Actualizar”, se llama al método setOnClickListener y se crea la clase dbhelper con la base de datos. Además, accede a ella en modo escritura, tal y como ya hemos hecho en otros archivos anteriormente:

```
//Al pulsar el boton actualizar
      actualizar.setOnClickListener(new View.OnClickListener() {

          @Override
          public void onClick(View arg0) {
             //creo la conexion y acceso en modo escritura
             DBHelper dbhelper = new DBHelper(getApplicationContext(),
"DBMascotas", null, 1);
             db = dbhelper.getWritableDatabase();
```

Seguidamente, agrego (mediante .getText().toString() y .getSelectedItem().toString()) a las variables los nuevos valores que el usuario modifique en la actividad DetailsActivity.java:

```
String nombre1 = ednombre.getText().toString();
             String fecha1 = _fecha.getText().toString();
             String sexo1 = edspinner1.getSelectedItem().toString();
             String peso1 = edpeso.getText().toString();
             String chip1 = edchip.getText().toString();
             String raza1 = edraza.getText().toString();
             String pedigree1 = edspinner2.getSelectedItem().toString();
             String propiet1 = edpropiet.getText().toString();
             String alergias1 = edalergias.getText().toString();
             String vacunas1 = edvacunas.getText().toString();
             String tratamientos1 = edtratamientos.getText().toString();
             String despint1 = eddespint.getText().toString();
             String despext1 = eddespext.getText().toString();
             String masinf1 = edmasinf.getText().toString();
```

El Statement creado en el archivo DBHelper.java (InsertarMascota) no sirve para actualizar los datos de la tabla, ya que éste solo puede modificar la tabla y no los valores que hay dentro de la tabla.

Por ello, creo la variable sql y agrego el código *UPDATE* para actualizar los campos de la base de datos con los nuevos campos nombrados como String en el paso anterior. Esto se hace mediante el id de cada mascota (este código es un poco tedioso por todos los caracteres que requiere. Con el método InsertarMascota nos ahorramos esto al momento de agregar una mascota nueva):

```
String sql = "UPDATE mascotas SET
nombre1='"+nombre1+"',fecha1='"+fecha1+"',sexo1='"+sexo1+"',peso1='"+peso1
+"',chip1='"+chip1+"',raza1='"+raza1+"',pedigree1='"+pedigree1+"',propiet1='"+pr
```

```
opiet1+"',alergias1='"+alergias1+"',vacunas1='"+vacunas1+"',tratamientos1='"+tr
atamientos1+"',despint1='"+despint1+"',despext1='"+despext1+"',masinf1='"+ma
sinf1+"' WHERE id="+id;
```

Si los valores de la base de datos son distintos a nulo (es decir, funcionan correctamente), se ejecuta la actualización mediante .execSQL y se muestra un Toast al usuario. Además, se crea un Intent(in) que nos lleve automáticamente a la lista con las mascotas actualizadas con las modificaciones que ha hecho el usuario en la actividad DetailsActivity.java:

```
if (db != null) {
                db.execSQL(sql);
                Toast.makeText(DetailsActivity.this, "Mascota actualizada
correctamente", Toast.LENGTH_SHORT).show();
                //Vuelvo a la lista para ver los cambios
                Intent in = new Intent(getApplicationContext(), PetsActivity.class);
                startActivity(in);
```

En caso de que surgiera algún problema actualizando los datos, se lanzará otro *Toast* al usuario:

```
} else {
                Toast.makeText(DetailsActivity.this, "Error al actualizar",
Toast.LENGTH_SHORT).show();
            }
        }
    });
    //FIN PARA ACTUALIZAR LA INFORMACION DE LA MASCOTA
```

2.5.7 INFACTIVITY

El archivo InfActivity.java es el que complementa al archivo activity_inf.xml.

El archivo InfActivity.java es el siguiente:

```
package com.example.doggycontrol;

import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageButton;

public class InfActivity extends AppCompatActivity {

    ImageButton informacionboton, recordatoriosboton;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_inf);


        recordatoriosboton = (ImageButton)
findViewById(R.id.idrecordatoriosbutton);
        recordatoriosboton.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent gotoreminder = new Intent(InfActivity.this,
AlarmaActivity.class);

                startActivity(gotoreminder);
            }
        });

        informacionboton = (ImageButton)
findViewById(R.id.idenfermedadesbutton);
        informacionboton.setOnClickListener(new View.OnClickListener()
{
            @Override
            public void onClick(View view) {
                Intent gotoillness = new Intent(InfActivity.this,
IllnessActivity.class);

                startActivity(gotoillness);
            }
        });

    }
}
```

Lo primero que hacemos en este archivo es definir las variables con las que vamos a trabajar:

```
ImageButton informacionboton, recordatoriosboton;
```

Seguidamente, se enlazan estas variables con sus respectivos elementos en su archivo activity_inf.xml y desarrollamos las acciones al pulsar en cada ImageButton.

Finalmente, se lanza en cada onClickListener un Intent(gotoreminder, gotoillness) para que se nos abra la respectiva actividad deseada(AlarmaActivity.java, IllnessActivity.java:

```
recordatoriosboton = (ImageButton) findViewById(R.id.idrecordatoriosbutton);
    recordatoriosboton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent gotoreminder = new Intent(InfActivity.this, AlarmaActivity.class);

            startActivity(gotoreminder);
```

```
        }
    });


informacionboton = (ImageButton) findViewById(R.id.idenfermedadesbutton);
    informacionboton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent gotoillness = new Intent(InfActivity.this, IllnessActivity.class);

            startActivity(gotoillness);
        }
    });
```

2.5.8 ALARMAACTIVITY

El archivo AlarmaActivity.java es el que complementa al archivo activity_alarma.xml.

Trabaja junto con el archivo Workmanagernoti.java para hacer posible la recepción de notificaciones al usuario.

El archivo AlarmaActivity.java es el siguiente:

```
package com.example.doggycontrol;

import android.app.DatePickerDialog;
import android.app.TimePickerDialog;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.DatePicker;
import android.widget.EditText;
import android.widget.TimePicker;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import androidx.work.Data;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.UUID;

public class AlarmaActivity extends AppCompatActivity {

    EditText _titulo, _descripcion;
    Button guardar;

//HACE FALTA Cactual porque se trabaja con dos variables de fecha y
tiempo (actual y el elegido para la notificacion)
```

```
    //PARA LA FECHA
    private int day,month,year;
    EditText fecha3;
    Calendar C = Calendar.getInstance(); //para seleccionar la fecha
deseada
    Calendar Cactual = Calendar.getInstance(); //para detectar la
fecha/hora actual del dispositivo
    DatePickerDialog.OnDateSetListener onDateSetListener;
    //FIN PARA LA FECHA

    //PARA SELECCIONAR LA HORA
    EditText hora3;
    private int min, hour;
   //tambien se usa Cactual
    //FIN PARA SELECCIONAR LA HORA

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_alarma);

        //PARA LA FECHA
        fecha3 = (EditText) findViewById(R.id.iddate2);
        fecha3.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {

                month = Cactual.get(Calendar.MONTH); //para que al
abrir se ponga en la fecha actual
                day = Cactual.get(Calendar.DAY_OF_MONTH);
                year = Cactual.get(Calendar.YEAR);

                DatePickerDialog datePickerDialog = new
DatePickerDialog(view.getContext(), new
DatePickerDialog.OnDateSetListener() {
                    @Override
                    public void onDateSet(DatePicker view, int y, int
m, int d) {
                        C.set(Calendar.DAY_OF_MONTH,d); //seteo en C
el dia elegido en el picker (sirve para Alerrtime)
                        C.set(Calendar.MONTH,m);//el mes seleccionado
en el calendario de C será m
                        C.set(Calendar.YEAR,y);

                        m = m+1; //se suma 1 al mes pq los meses se
indexan desde 0

                       fecha3.setText(d+"/"+m+"/"+y);//lo seteo
                    }
                },year, month, day);
                datePickerDialog.show();
            }
        });

        //FIN PARA LA FECHA



        //PARA LA HORA
```

```
        //agrego Cactual para que C no cambie
        hora3 = (EditText) findViewById(R.id.idhour2);
        hora3.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                hour = Cactual.get(Calendar.HOUR_OF_DAY); //para que
al abrir ponga la hora actual
                min = Cactual.get(Calendar.MINUTE); //para que al
abrir ponga los minutos actuales


                TimePickerDialog timePickerDialog = new
TimePickerDialog(view.getContext(), new
TimePickerDialog.OnTimeSetListener() {
                    @Override
                    public void onTimeSet(TimePicker view, int h, int
m) {
                        C.set(Calendar.HOUR_OF_DAY, h); //seteo en C
la hora elegida en el picker
                        C.set(Calendar.MINUTE,m); //seteo en C los
minutos elegidos en picker(sirve para el Alerttime)

                        hora3.setText(String.format("%02d:%02d",
h,m)); //con este codigo ponemos que si el numero es <9, le coloca un
0 delante
                    }
                },hour, min, true); //pongo true para que se vea en
24h
                timePickerDialog.show();
            }
        });
        //FIN PARA LA HORA




        guardar = findViewById(R.id.btn_guardar);
        guardar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String tag = generateKey(); //genera un tag random
                Long Alerttime = C.getTimeInMillis() -
System.currentTimeMillis(); //fecha seleccionada del calendar C menos
la hora del sistema(en ms)
                int random = (int)(Math.random() * 50 + 1); //esta
línea es simplemente para generar un valor aleatorio de un rango en
especifico. Numero entero random,n° aleatorio entre 0 y 1, que se
ultiplica por 50 y se le suma 1.


                _titulo = findViewById(R.id.idtitulo);
                String titulo = _titulo.getText().toString(); //lo
convierto en String para meterlo en el Data
                _descripcion = findViewById(R.id.iddescripcion);
                String descripcion =
_descripcion.getText().toString();//lo convierto en String = q
titulo1;


                Data data = GuardarData(titulo, descripcion, random);
                Workmanagernoti.GuardarNoti(Alerttime, data, tag);
```

```
                Toast.makeText(AlarmaActivity.this, "Recordatorio
agregado", Toast.LENGTH_SHORT).show();
            }
        });

    } //fin del onCreate



    private String generateKey(){
        return UUID.randomUUID().toString();
    }

    //lo siguiente es para guardar la informacion para llevarla a
nuestro Workmanagernoti
    private Data GuardarData (String titulo, String descripcion, int
id_noti){

        return new Data.Builder()
                .putString("titulo",titulo)
                .putString("descripcion",descripcion)
                .putInt("id_noti",id_noti).build();
    }


}
```

La selección de fecha mediante el diálogo DatePicker se realiza de la misma manera que hemos visto en las actividades AddActivity.java y DetailsActivity.java. Así mismo, el diálogo de selección de hora TimePicker funciona igual que el DatePicker, cambiando sus variables de *year,month,day* a *hour,min*. Por ello y con objetivo de no repetir información, no comentaremos estos códigos.

La única modificación a recalcar es que en esta actividad trabajamos con dos calendarios (C y Cactual), ya que un calendario (Cactual) almacenará la fecha/hora actual del dispositivo y el otro calendario (C) almacenará la fecha/hora elegida por el usuario para recibir la notificación.

```
public void onClick(View view) {

        month = Cactual.get(Calendar.MONTH); //para que al abrir se ponga en
la fecha actual
        day = Cactual.get(Calendar.DAY_OF_MONTH);
        year = Cactual.get(Calendar.YEAR);

        DatePickerDialog datePickerDialog = new
DatePickerDialog(view.getContext(), new DatePickerDialog.OnDateSetListener() {
            @Override
            public void onDateSet(DatePicker view, int y, int m, int d) {
```

```
                C.set(Calendar.DAY_OF_MONTH,d); //seteo en C el dia elegido en
el picker (sirve para Alerrtime)
                C.set(Calendar.MONTH,m);//el mes seleccionado en el calendario
de C será m
                C.set(Calendar.YEAR,y);
-------------------------arriba fecha; abajo hora-------------------------------
        public void onClick(View view) {
            hour = Cactual.get(Calendar.HOUR_OF_DAY); //para que al abrir ponga
la hora actual
            min = Cactual.get(Calendar.MINUTE); //para que al abrir ponga los
minutos actuales


            TimePickerDialog timePickerDialog = new
TimePickerDialog(view.getContext(), new TimePickerDialog.OnTimeSetListener()
{
                @Override
                public void onTimeSet(TimePicker view, int h, int m) {
                    C.set(Calendar.HOUR_OF_DAY, h); //seteo en C la hora elegida en
el picker
                    C.set(Calendar.MINUTE,m); //seteo en C los minutos elegidos en
picker(sirve para el Alerttime)
```

Esto es importante ya que, como veremos más adelante, cuando agreguemos al código la duración de la notificación (que hace referencia al tiempo que ha de pasar para que el usuario reciba la notificación), se restará la fecha/hora seleccionada por el usuario y almacenada en C menos la fecha/hora actual del dispositivo.

Con la fecha y hora ya seleccionadas, pasamos a explicar el resto del código. Se agregan las variables a utilizar:

```
EditText _titulo, _descripcion;
   Button guardar;
```

Se enlaza la variable guardar con su elemento en el layout y se agrega el método onClick para el botón de guardar:

```
guardar = findViewById(R.id.btn_guardar);
      guardar.setOnClickListener(new View.OnClickListener() {
```

Dentro de este método:

1-Agregamos una String nueva (tag), que será igual a una String random programada unas líneas más abajo:

```
String tag = generateKey(); //genera un tag random
...
```

```
…
…
private String generateKey(){
      return UUID.randomUUID().toString();
   }
```

2-Creamos la clase Long que almacenará la duración en la que queremos que se notifique al usuario. Como lo he comentado anteriormente, esto se consigue restando la fecha/hora seleccionada por el usuario y almacenada en C menos la fecha/hora actual del dispositivo:

```
            Long Alerttime = C.getTimeInMillis() - System.currentTimeMillis();
//fecha seleccionada del calendar C menos la hora del sistema(en ms)
```

3-Se enlazan las variables de título y descripción con sus respectivos elementos en el layout y se coge su contenido (lo que haya escrito el usuario) para convertirlo en String:

```
_titulo = findViewById(R.id.idtitulo);
            String titulo = _titulo.getText().toString(); //lo convierto en String para
meterlo en el Data
            _descripcion = findViewById(R.id.iddescripcion);
            String descripcion = _descripcion.getText().toString();//lo convierto en
String = q titulo1;
```

4-Se meten estas String en la variable Data (data) mediante el proceso situado unas líneas más abajo “GuardarData”, recurriendo a un constructor que coloca los datos dentro de este objeto:

```
Data data = GuardarData(titulo, descripcion, random);
…
…
…
   //lo siguiente es para guardar la informacion para llevarla a nuestro
Workmanagernoti
   private Data GuardarData (String titulo, String descripcion, int id_noti){
      return new Data.Builder()
            .putString("titulo",titulo)
            .putString("descripcion",descripcion)
            .putInt("id_noti",id_noti).build();
   }
```

5-Finalmente, lo metemos todo dentro del método GuardarNoti (enviándolo a la actividad Workmanagernoti.java que veremos en el siguiente apartado) y se enviará al usuario un *Toast* informando de que el recordatorio ha sido

agregado:

```
Workmanagernoti.GuardarNoti(Alerttime, data, tag);

            Toast.makeText(AlarmaActivity.this, "Recordatorio agregado",
Toast.LENGTH_SHORT).show();
        }
    });
```

2.5.9 WORKMANAGERNOTI

El archivo Workmanagernoti.java no se complementa con ningún archivo xml, sino que se encarga de recoger los datos de la notificación (y que previamente se han tratado en AlarmaActivity.java) y la lanza.

WorkManager facilita la programación de tareas asíncronas que se deben ejecutar incluso si la aplicación se cierra.
En vez de usar la API WorkManager se podría haber usado AlarmManager. Sin embargo, cuando se usa ésta última y el dispositivo se apaga/enciende/reinicia, las alarmas/notificaciones/recordatorios creados se eliminan.

Lo primero que hacemos es, en el archivo build.gradle(module:DoggyControl.app) y en el apartado de “dependencies”, agregar la siguiente línea:

```
//para trabajar con WorkManager
implementation 'androidx.work:work-runtime:2.3.0-beta02'
//fin para trabajar con WorkManager
```

Esta dependencia es la que nos permitirá utilizar WorkManager en nuestro proyecto, específicamente, en el archivo Workmanagernoti.java, permitiéndonos crear la clase extendida de “Worker” que veremos en el código a continuación.

El archivo Workmanagernoti.java es el siguiente:

```
package com.example.doggycontrol;

import android.app.NotificationChannel;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
```

```
import android.os.Build;
import androidx.annotation.NonNull;
import androidx.core.app.NotificationCompat;
import androidx.work.Data;
import androidx.work.OneTimeWorkRequest;
import androidx.work.WorkManager;
import androidx.work.Worker;
import androidx.work.WorkerParameters;
import java.util.Random;
import java.util.concurrent.TimeUnit;


public class Workmanagernoti extends Worker {

    public Workmanagernoti(@NonNull Context context, @NonNull
WorkerParameters workerParams) {
        super(context, workerParams);
    }

    public static void GuardarNoti(Long duracion, Data data,String
tag) {
        OneTimeWorkRequest noti = new
OneTimeWorkRequest.Builder(Workmanagernoti.class)
                .setInitialDelay(duracion,
TimeUnit.MILLISECONDS).addTag(tag)
                .setInputData(data).build();

        WorkManager instance = WorkManager.getInstance();
        instance.enqueue(noti); //le pasamos la inf desde el
alarmaactivity hasta el servicio workmanagernoti

    }

    @NonNull
    @Override
    public Result doWork() { //aquí dentro creamos todas nuestras
actividades

        String titulo = getInputData().getString("titulo");
        String descripcion = getInputData().getString("descripcion");

        oreo(titulo, descripcion);

        return Result.success();
    }

    private void oreo(String t, String d) {
        String id = "message";
        NotificationManager nm = (NotificationManager)
getApplicationContext().getSystemService(Context.NOTIFICATION_SERVICE)
;
        NotificationCompat.Builder builder = new
NotificationCompat.Builder(getApplicationContext(),id);

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            NotificationChannel nc = new NotificationChannel(id,
"nuevo", NotificationManager.IMPORTANCE_HIGH);
            nc.setDescription("Notificacion FCM");
            nc.setShowBadge(true);
            assert nm != null;
            nm.createNotificationChannel(nc);
```

```
        }

        Intent intent = new Intent(getApplicationContext(),
AlarmaActivity.class);

        intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP |
Intent.FLAG_ACTIVITY_SINGLE_TOP);
        PendingIntent pendingIntent =
PendingIntent.getActivity(getApplicationContext(), 0, intent,
PendingIntent.FLAG_ONE_SHOT);

        builder.setAutoCancel(true) //al clickar en la notif se borra
                .setWhen(System.currentTimeMillis())
                .setContentTitle(t)
                .setTicker("Nueva notificacion")
                .setSmallIcon(R.mipmap.ic_launcher)
                .setContentText(d)
                .setContentIntent(pendingIntent)
                .setContentInfo("nuevo");

        Random random = new Random();
        int idNotify = random.nextInt(8000);

        assert nm != null;
        nm.notify(idNotify,builder.build());
    }
}
```

Lo primero que hacemos, es extender la clase Workmanagernoti a “Worker”. Aquí estamos utilizando la dependencia que acabamos de añadir al build.gradle:

```
public class Workmanagernoti extends Worker {
```

Creamos el método GuardarNoti, el cual recibirá la información que el usuario ha pasado desde la pantalla AlarmaActivity.java y creará la instancia para que entre en funcionamiento con WorkManager. Este método necesitará para funcionar la duración (que hará referencia al tiempo que ha de pasar para que se envíe la notificación al usuario), data (que hace referencia a los datos que ingresa el usuario y que mostrará la notificación) y un tag (este tag actúa como la “etiqueta” del recordatorio).

Creamos un OneTimeWorkRequest (noti) para que la notificación se envíe una única vez, que será la seleccionada por el usuario en fecha y hora, y le pasamos la duración (mediante .setInitialDelay), data (mediante .setInputData) y el tag (.addTag). Finalmente, creamos el WorkManager (instance), que almacenará toda es información (noti):

```
public static void GuardarNoti(Long duracion, Data data,String tag) {
    OneTimeWorkRequest noti = new
```

```
OneTimeWorkRequest.Builder(Workmanagernoti.class)
            .setInitialDelay(duracion, TimeUnit.MILLISECONDS).addTag(tag)
            .setInputData(data).build();

      WorkManager instance = WorkManager.getInstance();
      instance.enqueue(noti); //le pasamos la inf desde el alarmaactivity hasta el
servicio workmanagernoti
 }
```

Seguidamente, recibimos el resultado a través de “do Work”, dentro del cual se crean nuestras actividades, recogiendo como String el título y la descripción de la notificación. Tras esto, llamamos al método que se encargará de crear la notificación (el cual hemos llamado “oreo”) y agregamos un “return success”, puesto que el resultado se da cuando nuestra tarea se haya llevado a cabo de manera satisfactoria:

```
   public Result doWork() { //aquí dentro creamos todas nuestras actividades

      String titulo = getInputData().getString("titulo");
      String descripcion = getInputData().getString("descripcion");

      oreo(titulo, descripcion);

      return Result.success();
   }
```

Al método oreo se le agrega las String de titulo y descripción y, dentro del método, se crea un objeto(nm) de clase NotificationManager, (que nos ayudará a interactuar con las notificaciones en segundo plano) y su respectivo constructor:

```
private void oreo(String t, String d) {
      String id = "message";
      NotificationManager nm = (NotificationManager)
getApplicationContext().getSystemService(Context.NOTIFICATION_SERVICE);
      NotificationCompat.Builder builder = new
NotificationCompat.Builder(getApplicationContext(),id);
```

Seguidamente, pasamos a crear el canal de notificación, mediante el cual se configura el comportamiento visual y auditivo que se aplicará a las notificaciones.

En nuestro caso, agregamos una importancia alta (para que la notificación ocupe un espacio en la pantalla durante unos segundos), una descripción FCM (*Firebase Cloud Messaging*, que permite enviar mensajes de notificación de forma segura) y se establece, mediante *.setShowBadge*, que las notificaciones

del canal aparezcan como íconos de aplicación.

Finalmente, se crea (con todas estas características) el canal de notificación:

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        NotificationChannel nc = new NotificationChannel(id, "nuevo",
NotificationManager.IMPORTANCE_HIGH);
        nc.setDescription("Notificacion FCM");
        nc.setShowBadge(true);
        assert nm != null;
        nm.createNotificationChannel(nc);
    }
```

Seguidamente, usaremos marcas de Intent para definir cómo se asocian las notificaciones y su comportamiento. Para ello, creamos el Intent(intent) y le agregamos las marcas *FLAG_ACTIVITY_CLEAR_TOP* y *FLAG_ACTIVITY_SINGLE_TOP*, que harán que cada vez que se agrega una nueva tarea, ésta se ponga detrás de la tarea ya existente. De esta manera, podremos enviar cuantas notificaciones queramos y todas llegarán en el tiempo específicado, pues las más nuevas no le quitarán jerarquía a las más antiguas:

```
    Intent intent = new Intent(getApplicationContext(), AlarmaActivity.class);
    intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP |
Intent.FLAG_ACTIVITY_SINGLE_TOP);
```

Recurrimos a un objeto PendingIntent para especificar una acción (actividad, en este caso) a tomar en el futuro. Le permite pasar una Intención futura y que la aplicación ejecute esa intención como si tuviera los permisos. Por lo tanto, esto significa, básicamente, que su aplicación en segundo plano no tiene que estar siempre en ejecución para que este proceso se lleve a cabo. Se le agrega la marca *FLAG_ONE_SHOT* para que PendingIntent solo funcione una vez cuando se le llame:

```
    PendingIntent pendingIntent =
PendingIntent.getActivity(getApplicationContext(), 0, intent,
PendingIntent.FLAG_ONE_SHOT);
```

Seguidamente, se agregan las características de la notificación al constructor (.setAutoCancel para que al pulsar en la notificación se elimine, .setContentTitle(t) para que el título de la notificación sea la variable t (que hemos sustituido en doWork por “titulo”), .setContentText(d) para que la descripción de la notificación sea la variable d (que hemos sustituido en doWork por “descripcion”), el icono de la notificación, etc:

```
builder.setAutoCancel(true) //al clickar en la notif se borra
        .setWhen(System.currentTimeMillis())
        .setContentTitle(t)
        .setTicker("Nueva notificacion")
        .setSmallIcon(R.mipmap.ic_launcher)
        .setContentText(d)
        .setContentIntent(pendingIntent)
        .setContentInfo("nuevo");
```

Creamos un id de notificación (idNotify) aleatorio para cada notificación (random). Esto hará que se pueda trabajar con varias notificaciones sin que ninguna sobreescriba a otra, ya que cada una tendrá su idNotify asignado aleatoriamente.

Finalmente, se agrega esto al NotificationManager(nm) creado al principio del método oreo, bajo la acción de notificar (nm.notify):

```
Random random = new Random();
    int idNotify = random.nextInt(8000);

    assert nm != null;
    nm.notify(idNotify,builder.build());
}
```

2.5.10 ILLNESSACTIVITY

El archivo IllnessActivity.java es el que complementa al archivo activity_illness.xml.

El archivo IllnessActivity.java es el siguiente:

```
package com.example.doggycontrol;

import androidx.appcompat.app.AppCompatActivity;

import android.app.AlertDialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.util.Log;
import android.view.View;

public class IllnessActivity extends AppCompatActivity {
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_illness);
    }

    //Dialog de la primera enfermedad

    public void ShowDialog1(View view) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setCancelable(false);
        builder.setTitle("Borreliosis canina o enfermedad de Lyme");
        builder.setMessage("-Descripción: Enfermedad causada por una
bacteria que se transmite mediante la picadura de una garrapata
infectada. Un análisis de sangre probará si tu perro sufre la
enfermedad. Los collares antigarrapatas son altamente efectivos contra
estos parásitos. También es conveniente revisar el pelo de tu perro
para comprobar si tiene alguna garrapata y poder extraérsela. \n\n" +
                "-Síntomas: fiebre, escalofríos, falta de energía,
inflamación muscular y de las articulaciones y posible cojera, arqueo
de espalda…\n\n" +
                "NO SE CONTAGIA DE PERROS A HUMANOS.");
        builder.setPositiveButton("Ok", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int id) {
                Log.d("Mensaje", "Se canceló la acción");
            }
        })
                .show();

    }


    //Dialog de la segunda enfermedad

    public void ShowDialog2(View view) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setCancelable(false);
        builder.setTitle("Brucelosis");
        builder.setMessage("-Descripción: enfermedad causada por una
bacteria que daña el sistema reproductivo, por lo que es causa de
esterilidad y aborto espontáneo en los perros. Aunque en un 80% de los
casos se elimina mediante administración de antibióticos, se trata de
una enfermedad difícil de erradicar.  \n\n" +
                "-Síntomas: aborto espontáneo o esterilidad, cachorros
que fallecen al poco tiempo de nacer y ganglios linfáticos o
testículos agrandados.\n\n" +
                "SÍ SE CONTAGIA DE PERROS A HUMANOS.");
        builder.setPositiveButton("Ok", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int id) {
                Log.d("Mensaje", "Se canceló la acción");
            }
        })
                .show();

    }
```

```
    public void ShowDialog3(View view) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setCancelable(false);
        builder.setTitle("Cáncer");
        builder.setMessage("-Descripción: Un cáncer es una afección en
la que se produce una rápida división celular. Las células se
reproducen a un ritmo excesivo y pueden extenderse por metástasis a
otras zonas del cuerpo, llegando a provocar incluso la muerte del
perro. La eficacia de los tratamientos depende del diagnóstico precoz.
Acude a tu veterinario para que lleve a cabo el tratamiento más
adecuado para tu perro. La mayoría de los cánceres son detectables
mediante examen físico, por lo que una exploración física anual puede
ayudar a prevenirlos de manera efectiva.\n\n" +
                "-Síntomas: cansancio, pérdida de peso, falta de
apetito, caída del estado anímico, dolor, alopecia, bultos extraños,
inflamaciones, sangrado…\n\n" +
                "NO SE CONTAGIA DE PERROS A HUMANOS.");
        builder.setPositiveButton("Ok", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int id) {
                Log.d("Mensaje", "Se canceló la acción");
            }
        })
                .show();

    }



    public void ShowDialog4(View view) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setCancelable(false);
        builder.setTitle("Conjuntivitis");
        builder.setMessage("-Descripción: Se trata de una inflamación
de la membrana conjuntiva y es un problema muy común en los perros. En
casos leves, puede ser tratada en casa mediante administración de un
colirio. No obstante, conviene consultar con el veterinario por si
necesitara algún tratamiento más intensivo.\n\n" +
                "-Síntomas: enrojecimiento, hinchazón, picazón y
secreción ocular, parpadeo continuo…\n\n" +
                "SÍ SE CONTAGIA DE PERROS A HUMANOS, pero es poco
probable y solo si es producida por infección vírica.");
        builder.setPositiveButton("Ok", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int id) {
                Log.d("Mensaje", "Se canceló la acción");
            }
        })
                .show();

    }



    public void ShowDialog5(View view) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setCancelable(false);
```

```
        builder.setTitle("Gastritis aguda (vómito agudo)");
        builder.setMessage("-Descripción: La gastritis aguda es una
inflamación del revestimiento gastrointestinal en el estómago causada
generalmente por la ingesta de sustancias contaminadas o no preparadas
para ser digeridas. Generalmente desaparece sola en 24-48 horas, pero
en caso de que agrave, acude a tu veterinario.\n\n" +
                "-Síntomas: diarrea y vómitos excesivos (puede incluir
sangre), heces negras y alquitranadas, dolor abdominal…\n\n" +
                "NO SE CONTAGIA DE PERROS A HUMANOS.");
        builder.setPositiveButton("Ok", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int id) {
                Log.d("Mensaje", "Se canceló la acción");
            }
        })
                .show();

    }



    public void ShowDialog6(View view) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setCancelable(false);
        builder.setTitle("Hepatitis canina");
        builder.setMessage("-Descripción: Se trata de una enfermedad
muy contagiosa causada por un virus que se libera en las secreciones
corporales y que afecta, en especial, al hígado y a los riñones. La
vacunación es muy eficaz para prevenir esta enfermedad.\n\n" +
                "-Síntomas: fiebre, vómitos, diarrea, falta de
energía, hígado o amígdalas agrandadas, dolor abdominal, moretones,
puntos rojos en la piel, ganglios linfáticos inflamados, letargo…\n\n"
+
                "NO SE CONTAGIA DE PERROS A HUMANOS.");
        builder.setPositiveButton("Ok", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int id) {
                Log.d("Mensaje", "Se canceló la acción");
            }
        })
                .show();

    }



    public void ShowDialog7(View view) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setCancelable(false);
        builder.setTitle("Leishmaniosis");
        builder.setMessage("-Descripción: La leishmaniosis es una
enfermedad parasitaria grave, incluso mortal, causada por un protozoo.
Esta enfermedad se transmite a través de la picadura de un mosquito
llamado Phlebotomus. Si el perro no recibe tratamiento, la enfermedad
puede ser mortal, por lo que, si crees que puede padecer la
enfermedad, debes acudir inmediatamente a tu veterinario. La
enfermedad se puede prevenir mediante la vacunación. Si tu perro está
```

```
bien inmunizado, aunque sea infectado con el parásito, podrá convivir
con él sin que le produzca grandes complicaciones. Los collares
antiparasitarios repelentes de mosquitos también son un método eficaz
de prevención.\n\n" +
                "-Síntomas: patologías dermatológicas, atrofia
muscular, fiebre, párpados inflamados, fatiga, cojera, alopecia en
zonas determinadas…\n\n" +
                "NO SE CONTAGIA DE PERROS A HUMANOS.");
        builder.setPositiveButton("Ok", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int id) {
                Log.d("Mensaje", "Se canceló la acción");
            }
        })
                .show();

    }



    public void ShowDialog8(View view) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setCancelable(false);
        builder.setTitle("Leptospirosis");
        builder.setMessage("-Descripción: Enfermedad causada por una
bacteria que se propaga a través de la orina, normalmente cuando ésta
se encuentra en agua estancada, la bacteria penetra en el cuerpo del
perro a través de una herida en la piel. \n\n" +
                "-Síntomas: fiebre, vómitos, diarrea, mucosas
amarillentas, aumento de volumen de orina, debilidad, pérdida de
apetito, trastornos neurológicos…\n\n" +
                "SÍ SE CONTAGIA DE PERROS A HUMANOS, pero es poco
frecuente.");
        builder.setPositiveButton("Ok", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int id) {
                Log.d("Mensaje", "Se canceló la acción");
            }
        })
                .show();

    }



    public void ShowDialog9(View view) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setCancelable(false);
        builder.setTitle("Mastitis");
        builder.setMessage("-Descripción: enfermedad de origen
bacteriano que causa una inflamación en las glándulas mamarias que
afecta a la perra y, posiblemente, a sus cachorros. Suele tener lugar
en el periodo post-parto.\n\n" +
                "-Síntomas: hinchazón y dolor en el área mamaria,
pérdida de apetito, debilidad, letargo, infección, rechazo a los
cachorros…\n\n" +
                "NO SE CONTAGIA DE PERROS A HUMANOS.");
```

```
        builder.setPositiveButton("Ok", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int id) {
                Log.d("Mensaje", "Se canceló la acción");
            }
        })
                .show();

    }



    public void ShowDialog10(View view) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setCancelable(false);
        builder.setTitle("Moquillo");
        builder.setMessage("-Descripción: El moquillo es una
enfermedad muy contagiosa debida a un virus parecido al del sarampión
humano. La vacunación de tu perro le protegerá.\n\n" +
                "-Síntomas: fiebre, vómitos, diarrea, falta de
apetito, adelgazamiento, estornudos, tos, dificultad de respirar,
secreción nasal, conjuntivitis…\n\n" +
                "NO SE CONTAGIA DE PERROS A HUMANOS.");
        builder.setPositiveButton("Ok", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int id) {
                Log.d("Mensaje", "Se canceló la acción");
            }
        })
                .show();

    }



    public void ShowDialog11(View view) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setCancelable(false);
        builder.setTitle("Otitis");
        builder.setMessage("-Descripción: infección del conducto
auditivo que se da frecuentemente en razas de oreja larga y caída. Es
dolorosa y el tratamiento consiste en la limpieza y secado del
conducto auditivo, que debe realizarlo el veterinario. \n\n" +
                "-Síntomas: molestia y dolor en las orejas, mal olor,
fiebre, picor y rascado frecuente, sacudidas, secreciones
oscuras…\n\n" +
                "SÍ SE CONTAGIA DE PERROS A HUMANOS, pero es muy poco
frecuente.");
        builder.setPositiveButton("Ok", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int id) {
                Log.d("Mensaje", "Se canceló la acción");
            }
        })
                .show();

    }
```

```
    public void ShowDialog12(View view) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setCancelable(false);
        builder.setTitle("Parvovirus");
        builder.setMessage("-Descripción: El parvovirus es una
enfermedad vírica muy contagiosa. El virus ataca a las células que se
están reproduciendo, como las del tracto intestinal, y se propaga a
través de las heces o del contacto oral/nasal. La enfermedad requiere
atención veterinaria. La vacunación previene la mayoría de los casos
de infección.\n\n" +
                "-Síntomas: fiebre, vómitos, diarrea (suele ir
acompañada de hemorragia), debilidad, deshidratación, pérdida de
apetito y peso, dolor al palpar el abdomen, letargo…\n\n" +
                "NO SE CONTAGIA DE PERROS A HUMANOS.");
        builder.setPositiveButton("Ok", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int id) {
                Log.d("Mensaje", "Se canceló la acción");
            }
        })
                .show();

    }



    public void ShowDialog13(View view) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setCancelable(false);
        builder.setTitle("Rabia");
        builder.setMessage("-Descripción: La rabia es una enfermedad
mortal que se da en la mayoría de los animales de sangre caliente. Se
contagia a través de la saliva de los animales infectados, aunque el
principal foco de infección es el mordisco de un animal infectado. La
vacunación es altamente efectiva, por lo que se recomienda y en
algunas situaciones, se obliga (por ejemplo, al viajar), que se vacune
al perro contra esta peligrosa enfermedad.\n\n" +
                "-Síntomas: carácter asustadizo, irritado, inquieto,
brusco o de ataque, sensibilidad a la luz o a los ruidos, aumento de
salivación y dificultades al tragar, desorientación, tendencia a
morder, parálisis…\n\n" +
                "SÍ SE CONTAGIA DE PERROS A HUMANOS. El tratamiento de
rabia en perros está prohibido debido al elevado peligro de contagio a
personas.");
        builder.setPositiveButton("Ok", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int id) {
                Log.d("Mensaje", "Se canceló la acción");
            }
        })
                .show();

    }
```

```
    public void ShowDialog14(View view) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setCancelable(false);
        builder.setTitle("Sarna demodécica");
        builder.setMessage("-Descripción: enfermedad de la piel
causada por un ácaro. En número reducido se consideran fauna habitual
de la piel, pero el problema es cuando su número se descontrola,
debido a un fallo del sistema inmunológico del perro. Debes acudir a
tu veterinario, que le administrará el tratamiento adecuado.\n\n" +
                "-Síntomas: alopecia, engrosamiento de la piel,
comedones, escamas, hiperpigmentación, eritemas…\n\n" +
                "NO SE CONTAGIA DE PERROS A HUMANOS.");
        builder.setPositiveButton("Ok", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int id) {
                Log.d("Mensaje", "Se canceló la acción");
            }
        })
                .show();

    }



    public void ShowDialog15(View view) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setCancelable(false);
        builder.setTitle("Sarna sarcóptica o sarna común");
        builder.setMessage("-Descripción: enfermedad de la piel
causada por un ácaro. Este parásito vive durante todo su ciclo
biológico encima del animal.\n\n" +
                "-Síntomas: picor excesivo y, por consecuencia: caída
de pelo, costras, enrojecimiento…\n\n" +
                "SÍ SE CONTAGIA DE PERROS A HUMANOS, pero las
consecuencias son mínimas.");
        builder.setPositiveButton("Ok", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int id) {
                Log.d("Mensaje", "Se canceló la acción");
            }
        })
                .show();

    }



    public void ShowDialog16(View view) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setCancelable(false);
        builder.setTitle("Torsión o dilatación gástrica (vólvulo
gástrico)");
        builder.setMessage("-Descripción: enfermedad potencialmente
mortal de progresión rápida que suele asociarse a comidas copiosas y,
```

```
por consecuencia, una dilatación gástrica en la cual el estómago se
distiende con gas y líquido. A continuación, se produce el vólvulo,
que consiste en la torsión, es decir, el estómago rota sobre su eje
longitudinal, desplazando el píloro de su posición. Es muy importante
que acudas a tu veterinario para que pueda poner remedio a la
situación. A menudo es necesaria una cirugía.\n\n" +
                "-Síntomas: dolor abdominal, jadeos, mayor salivación,
rechazo de comida, pared abdominal tensa, vientre dilatado, arcadas,
pocas ganas de moverse, temblores…\n\n" +
                "NO SE CONTAGIA DE PERROS A HUMANOS.");
        builder.setPositiveButton("Ok", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int id) {
                Log.d("Mensaje", "Se canceló la acción");
            }
        })
                .show();

    }



    public void ShowDialog17(View view) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setCancelable(false);
        builder.setTitle("Toxoplasmosis");
        builder.setMessage("-Descripción: enfermedad provocada por un
protozoo. La mayoría de los perros se infectan por la ingestión de
alimentos que contienen estos parásitos o de heces de gato infectadas.
Para diagnosticar la enfermedad, es necesario hacer una serología.
\n\n" +
                "-Síntomas: vómitos, diarrea, temblores, debilidad,
falta de coordinación, mucosas amarillentas…\n\n" +
                "NO SE CONTAGIA DE PERROS A HUMANOS.");
        builder.setPositiveButton("Ok", new
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int id) {
                Log.d("Mensaje", "Se canceló la acción");
            }
        })
                .show();

    }


    public void ShowDialog18(View view) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setCancelable(false);
        builder.setTitle("Traqueobronquitis infecciosa o tos de las
perreras");
        builder.setMessage("-Descripción: enfermedad respiratoria
producida por diversos virus y bacterias que similar a la gripe
humana. Generalmente se trata de una enfermedad leve que se cura sola,
pero es muy contagiosa. En algunos casos, se puede complicar con
neumonía o bronquitis.\n\n" +
                "-Síntomas: tos seca, áspera y constante, arcadas,
fiebre e inflamación de la garganta, tráquea o pulmones.\n\n" +
                "NO SE CONTAGIA DE PERROS A HUMANOS.");
        builder.setPositiveButton("Ok", new
```

```
DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int id) {
                Log.d("Mensaje", "Se canceló la acción");
            }
        })
                .show();
    }
}
```

En este archivo se repite el mismo procedimiento 18 veces, ya que son 18 botones que muestran 18 diálogos, respectivamente. Lo único que cambia es el contenido del texto de cada diálogo.

Explicaré el primer diálogo, que muestra la información de la primera enfermedad.

Primero, llamo al método del diálogo mediante *ShowDialog1*. Dentro de éste llamamos a la construcción de un *AlertDialog.Builder*.

Se agrega el título (*builder.setTitle*) y la descripción del cuadro de diálogo (*builder.setMessage*). Esto es lo que se mostrará en el diálogo al pulsar sobre el botón y mediante el código *.show().*

También agregamos un botón “Ok” mediante *builder.setPositiveButton* que se mostrará dentro del cuadro y, al pulsarlo, se cerrará el cuadro de diálogo:

```
//Dialog de la primera enfermedad
   public void ShowDialog1(View view) {
      AlertDialog.Builder builder = new AlertDialog.Builder(this);
      builder.setCancelable(false);
      builder.setTitle("Borreliosis canina o enfermedad de Lyme");
      builder.setMessage("-Descripción: Enfermedad causada por una bacteria
que se transmite mediante la picadura de una garrapata infectada. Un análisis de
sangre probará si tu perro sufre la enfermedad. Los collares antigarrapatas son
altamente efectivos contra estos parásitos. También es conveniente revisar el
pelo de tu perro para comprobar si tiene alguna garrapata y poder extraérsela.
\n\n" +
            "-Síntomas: fiebre, escalofríos, falta de energía, inflamación muscular y
de las articulaciones y posible cojera, arqueo de espalda…\n\n" +
            "NO SE CONTAGIA DE PERROS A HUMANOS.");
      builder.setPositiveButton("Ok", new DialogInterface.OnClickListener() {
         @Override
         public void onClick(DialogInterface dialog, int id) {
            Log.d("Mensaje", "Se canceló la acción");
         }
      })
            .show();

   }
```

2.5.11 CONTACTACTIVITY

El archivo ContactActivity.java es el que complementa al archivo activity_contact.xml.

El archivo ContactActivity.java es el siguiente:

```
package com.example.doggycontrol;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.Toast;

public class ContactActivity extends AppCompatActivity {

    ImageButton correoboton;
    ImageButton telefonoboton;
    ImageButton whatsappboton;
    EditText numerotelef;

    //onCreate común para todos los ImageButton

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_contact);


//LANZAR EL CORREO
        correoboton = (ImageButton) findViewById(R.id.idcorreobutton);


        correoboton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent mailIntent = new Intent(Intent.ACTION_VIEW);
                Uri data = Uri.parse("mailto:?subject=" + "Asunto del
e-mail" + "&body=" + "Texto del e-mail" + "&to=" +
"abc123@gmail.com");
                mailIntent.setData(data);
                startActivity(Intent.createChooser(mailIntent,
"Enviando e-mail..."));

            }
        });


//LANZAR EL WHATSAPP
        whatsappboton = (ImageButton)
findViewById(R.id.idwhatsappbutton);
```

```
        whatsappboton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                try {
                    Intent intent =
getPackageManager().getLaunchIntentForPackage("com.whatsapp");
                    Intent whatsappIntent =
Intent.createChooser(intent, null);
                    startActivity(whatsappIntent);
                } catch (Exception e) {
                    Toast.makeText(ContactActivity.this, "WhatsApp no
está instalado", Toast.LENGTH_SHORT).show();
                }
            }

        });



//LLAMAR AL TELEFONO ESPECIFICADO
        telefonoboton = (ImageButton)
findViewById(R.id.idtelefonobutton);
        numerotelef = (EditText) findViewById(R.id.idnumero);

        telefonoboton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String number = numerotelef.getText().toString();

                if (!number.equals(""))
                {
                    Intent intent2 = new Intent(Intent.ACTION_DIAL);
                    intent2.setData(Uri.parse("tel:" + number));
                    startActivity(intent2);
                } else {
                    Toast.makeText(ContactActivity.this, "Introduce el
número de teléfono para continuar", Toast.LENGTH_SHORT).show();
                }
            }
        });
    }
}
```

Primero, agregamos las variables que utilizaremos:

```
ImageButton correoboton;
ImageButton telefonoboton;
ImageButton whatsappboton;
EditText numerotelef;
```

1. Para el correo

Enlazamos la variable con su respectivo elemento del layout y llamamos a setOnClickListener.

Añadimos el Intent(mailIntent) que lanzará el correo.

Añadimos los datos que queremos que aparezcan por defecto en cada campo cuando se lance el correo en formato Uri (data), mediante la función Uri.parse. Los agregamos al Intent(mailIntent) para que se muestren en el correo y, finalmente, agregamos el código para iniciar la actividad (startActivity) y para que el usuario reciba el mensaje “Enviando e-mail…”:

```
//LANZAR EL CORREO
      correoboton = (ImageButton) findViewById(R.id.idcorreobutton);
      correoboton.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View view) {
            Intent mailIntent = new Intent(Intent.ACTION_VIEW);
            Uri data = Uri.parse("mailto:?subject=" + "Asunto del e-mail" +
"&body=" + "Texto del e-mail" + "&to=" + "abc123@gmail.com");
            mailIntent.setData(data);
            startActivity(Intent.createChooser(mailIntent, "Enviando e-mail..."));
         }
      });
```

2. Para el WhatsApp

Enlazamos la variable con su respectivo elemento del layout y llamamos a setOnClickListener.

Se intentará, mediante un Intent(intent), acceder al Package de WhatsApp en el dispositivo. Si lo consigue, significa que la aplicación WhatsApp se encuentra en nuestro dispositivo y se lanzará otro Intent(whatsappIntent) que abrirá la aplicación. Si no lo consigue, pasará a mostrar un *Toast* al usuario con el mensaje “Whatsapp no está instalado”:

```
//LANZAR EL WHATSAPP
      whatsappboton = (ImageButton) findViewById(R.id.idwhatsappbutton);
      whatsappboton.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View view) {
            try {
               Intent intent =
getPackageManager().getLaunchIntentForPackage("com.whatsapp");
               Intent whatsappIntent = Intent.createChooser(intent, null);
               startActivity(whatsappIntent);
            } catch (Exception e) {
               Toast.makeText(ContactActivity.this, "WhatsApp no está instalado",
Toast.LENGTH_SHORT).show();
            }
         }
      });
```

3. Para llamar por teléfono

Enlazamos las variables (telefonoboton y numerotelef) con sus respectivos elementos del layout y llamamos a setOnClickListener.

Se crea como un String “number”, que se encargará de coger, mediante el método .getText().toString(), el número especificado en el EditText.

Si hay números en el campo de texto, se lanza un Intent(intent2) para acceder a la pantalla dial del dispositivo, pasando (mediante la variable “data”) a esta actividad el número que hemos recogido.

Si no hay números en el campo de texto mostrará un *Toast* al usuario con el mensaje “Introduce el número de teléfono para continuar”:

```
//LLAMAR AL TELEFONO ESPECIFICADO
        telefonoboton = (ImageButton) findViewById(R.id.idtelefonobutton);
        numerotelef = (EditText) findViewById(R.id.idnumero);
        telefonoboton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String number = numerotelef.getText().toString();
                if (!number.equals(""))
                {
                    Intent intent2 = new Intent(Intent.ACTION_DIAL);
                    intent2.setData(Uri.parse("tel:" + number));
                    startActivity(intent2);
                } else {
                    Toast.makeText(ContactActivity.this, "Introduce el número de
teléfono para continuar", Toast.LENGTH_SHORT).show();
                }
            }
        });
```

CAPÍTULO 3. RESULTADOS Y DISCUSIÓN

En este apartado, trataré de analizar objetivamente este proyecto y discutir los resultados obtenidos con el fin de proponer mejoras en los aspectos que considere necesarios.

Como comenté al principio, decidí desarrollar DoggyControl porque en mi entorno tratamos mucho con mascotas caninas y, de acuerdo a las necesidades que nos han surgido, he ido poco a poco diseñando esta aplicación y dándole forma.

Partiendo de la base de que mi conocimiento en el entorno de la programación era totalmente nulo, para poder trabajar con este proyecto he tenido que hacer frente a diferentes estructuras de programación que ni siquiera sabía que existían, desde las más simples como puede ser añadir un ScrollView a un archivo xml, hasta las más complejas como puede ser trabajar con SQLite, ListView o WorkManager.

Sin embargo, el tener que adquirir estos conocimientos para poder emplear correctamente cada uno de los métodos/elementos que hemos visto y que la aplicación funcionase de acuerdo a lo que tenía en mente, me resultó realmente gratificante. Y es aquí donde me doy cuenta de que no solamente he realizado un Trabajo de Fin de Grado que entregaré y lo dejaré en el olvido, sino que he adquirido unos conocimientos y una manera de empezar a comprender la programación que el grado de Ingeniería Mecánica no me proporcionaba y que estoy segura que sacaré provecho.

Aunque creo que el resultado obtenido se acerca mucho a la aplicación deseada, se me ocurren algunas ideas para mejorar la aplicación:

1. Almacenar las imágenes seleccionadas de la galería o tomadas con la cámara en la base de datos, que se muestre y que se permita editarla, al igual que el resto de datos.
2. Crear un único archivo xml y un único archivo java para añadir mascota

y editar/actualizar mascota. Ya que, de esta manera, los cambios futuros se podrían realizar en el mismo archivo en el que se agregan los datos iniciales de las mascotas.

3. Personalizar el ListView de manera que aparezca la imagen/fotografía de cada animal.
4. Poder realizar una copia de seguridad de todos los datos almacenados.
5. Posibilidad de compartir información de la mascota.

CAPÍTULO 4. CONCLUSIONES

Una vez damos por finalizada la redacción de este proyecto y la programación de nuestra aplicación, es momento de reflexionar acerca de los resultados y de lo que este proyecto ha supuesto para mí.

Si bien es verdad que en el mercado hay ciertas aplicaciones similares y con un nivel más elevado de programación, creo que el tener en mi entorno ciertas necesidades y poder cubrirlas me ha ido motivando en el camino. Además, el ir viendo los resultados de algo de lo que quizá he estado buscando información durante varios días es, indiscutiblemente, un gran aliciente para adentrarme más y más en la programación.

En conclusión, estoy encantada de haber elegido desarrollar DoggyControl como Trabajo Final de Grado y me siento realmente orgullosa y satisfecha de haber podido adquirir todos estos conocimientos. Aunque aún queda mucho por aprender, me siento completamente incentivada a adentrarme cada vez más en este mundo tan amplio que es la programación.

CAPÍTULO 5. BIBLIOGRAFÍA

Recopilación de las fuentes de información utilizadas para este proyecto:

YouTube → https://www.youtube.com/

Desarrolladores de Android → https://developer.android.com/

LCSI → http://lcsi.umh.es/androidstudio/

StackOverflow → https://stackoverflow.com/

Wikipedia → https://es.wikipedia.org/wiki/Wikipedia:Portada

Google Play (diferentes aplicaciones) → https://play.google.com/store

Blog de Ángel J. Vico → https://columna80.wordpress.com/

JavaTpoint → https://www.javatpoint.com/

Academia Android → https://academiaandroid.com/

SgOliver → https://www.sgoliver.net/blog/

Red Canina → https://www.redcanina.es/perros/sobre-cuidar-perros/

Curso Android Studio → http://cursoandroidstudio.blogspot.com/

CAPÍTULO 6. TABLA DE ILUSTRACIONES

Ilustración 1: Logo Android Apple Pie……6
Ilustración 2: Logo Android Banana Bread……7
Ilustración 3: Logo Android Cupcake……7
Ilustración 4: Logo Android Donut……7
Ilustración 5: Logo Android Eclair……8
Ilustración 6: Logo Android Froyo……8
Ilustración 7: Logo Android Gingerbread……8
Ilustración 8: Logo Android HoneyComb……8
Ilustración 9: Logo Android Ice Cream Sandwich……9
Ilustración 10: Logo Android Jelly Bean……9
Ilustración 11: Logo Android KitKat……9
Ilustración 12: Logo Android Lollipop……10
Ilustración 13: Logo Android Apple Marshmallow……10
Ilustración 14: Logo Android Nougat……10
Ilustración 15: Logo Android Oreo……11
Ilustración 16: Logo Android Android Pie……11
Ilustración 17: Versiones de Android más utilizadas. Marzo2021……12
Ilustración 18: Interfaz “Mis proyectos”. MIT App Inventor 2……13
Ilustración 19: Interfaz “Diseñador”. MIT App Inventor 2……15
Ilustración 20: Interfaz “Editor de bloques”. MIT App Inventor 2……16
Ilustración 21: Interfaz principal. Android Studio……18
Ilustración 22: Requisitos recomendados para el uso de Android Studio……19
Ilustración 23: Pantallas de la aplicación Diario de la mascota……21
Ilustración 24: Pantallas de la aplicación Cuidado de mi mascota……23
Ilustración 25: Pantalla principal de DoggyControl……24
Ilustración 26: Pantalla de Pets Activity de DoggyControl……25
Ilustración 27: Pantalla de Add Activity de DoggyControl……26
Ilustración 28: Pantalla de Details Activity de DoggyControl……27
Ilustración 29: Pantalla de Inf Activity de DoggyControl……28
Ilustración 30: Pantalla de Alarma Activity de DoggyControl……28
Ilustración 31: Pantalla de Illness Activity de DoggyControl……29

Ilustración 32: Pantalla de Contact Activity de DoggyControl……...30
Ilustración 33: Logo de DoggyControl……...31
Ilustración 34: Botón de “Recordatorios”……..31
Ilustración 35: Botón de “Contactar por correo”……..….31
Ilustración 36: Configuración inicial de DoggyControl en Android Studio…..33