

INSTITUTO DE EDUCACIÓN SECUNDARIA SERPIS

Diseño de una aplicación móvil para el cuidado de mascotas FidoFriend

Proyecto de Desarrollo de Aplicaciones Multiplataforma

Ciclo Formativo Desarrollo de Aplicaciones Multiplataforma
Departamento de informática

Autor: Buenaño Evans, Kevin

Tutor: Chover Miñana, María Ángeles

Curso: 2022/2023



- Resumen -

(Español)

Se realiza una aplicación móvil para el cuidado de las mascotas, en esta aplicación se busca poder realizar un seguimiento para el cuidado de mascotas, sería una herramienta para poder tener un resumen más rápido y comprensible del estado de la mascota y así poder planificar objetivos para cuidarlas. Servirá también para tener un historial de seguimiento de la salud de la mascota y poder obtener avisos y consejos antes de tiempo para no empeorar su salud.

En cuanto a los resultados se podrá seguir un seguimiento de la salud, la actividad, su historial de limpieza. Tendrá funciones para mejorar el rendimiento y salud de la mascota acorde a lo que elija el usuario. En cada sección Salud y Limpieza se tendrá un historial con la fecha y en el caso de la Salud también tendrá un texto en donde se podrá mostrar un resumen del diagnóstico del veterinario. La aplicación mostrará los datos que más atención se debe tener cuando no cumplen un mínimo para que tenga una buena salud la mascota. Además, se podrán crear notificaciones para que el usuario se planifique mejor la atención de sus mascotas.

(Valenciano)

Es realitza una aplicació mòbil per a la cura de les mascotes, en aquesta aplicació es busca poder realitzar un seguiment per a la cura de mascotes, seria una eina per a poder tindre un resum més ràpid i compressible de l'estat de la mascota i així poder planificar objectius per a cuidar-les. Servirà també per a tindre un historial de seguiment de la salut de la mascota i poder obtindre avisos i consells abans d'hora per a no empitjorar la seua salut.

Quant als resultats es podrà seguir un seguiment de la salut, l'activitat, el seu historial de neteja. Tindrà funcions per a millorar el rendiment i salut de la mascota concorde al que trie l'usuari. En cada secció Salut i Neteja es tindrà un historial amb la data i en el cas de la Salut també tindrà un text on es podrà mostrar un resum del diagnòstic del veterinari. L'aplicació mostrarà les dades que més atenció s'ha de tindre quan no compleixen un mínim perquè tinga una bona salut la mascota. A més, es podran crear notificaciones perquè l'usuari es planifique millor l'atenció de les seues mascotes.

(Inglés)

A mobile application is made for the care of pets, this application seeks to be able to track the care of pets, it would be a tool to be able to have a quick and understandable

summary of the state of the pet and thus be able to plan objectives to care for them. It will also serve to have a history of monitoring the health of the pet and to be able to obtain warnings and advice in advance so as not to worsen their health. In terms of results, it will be possible to track the pet's health, activity and cleaning history. It will have functions to improve the performance and health of the pet according to what the user chooses. In each Health and Cleanliness section there will be a history with the date and in the case of Health it will also have a text where a summary of the veterinarian's diagnosis can be shown. The application will show the data that should be taken into account when they do not meet the minimum requirements for the pet to be in good health. In addition, notifications can be created so that the user can better plan the care of their pets.

Contenido

Índice de imágenes.....	3
- Justificación -.....	4
- Herramientas utilizadas -.....	5
- Descripción del proyecto -.....	7
Análisis:.....	7
Diseño:.....	8
Prototipo interfaz:.....	12
Desarrollo:.....	15
Base de Datos:.....	16
Entidades:.....	16
Interfaz menú:.....	19
Interfaz mascota:.....	22
Pruebas:.....	26
Documentación:.....	27
Trabajos futuros:.....	27
Conclusiones:.....	29
Bibliografía y webgrafía:.....	29
Anexos:.....	31

Índice de imágenes

Tablas:

Tabla 1.....	4
--------------	---

Diagramas:

Diagrama 1 Diagrama E/R.....	8
Diagrama 2 Diagrama de Clases.....	9
Diagrama 3 Casos de uso.....	10

Prototipo aplicación:

Prototipo 1.....	9
Prototipo 2.....	10
Prototipo 3.....	11
Prototipo 4.....	12

Gráficos de la aplicación:

Grafo 1 Interfaz menú.....	21
Grafo 2 Interfaz añadir mascota.....	22
Grafo 3 Interfaz menú mascota.....	23

Interfaz aplicación:

Ilustración 1 Ventana información.....	23
Ilustración 2 Modificar mascota.....	24

Código:

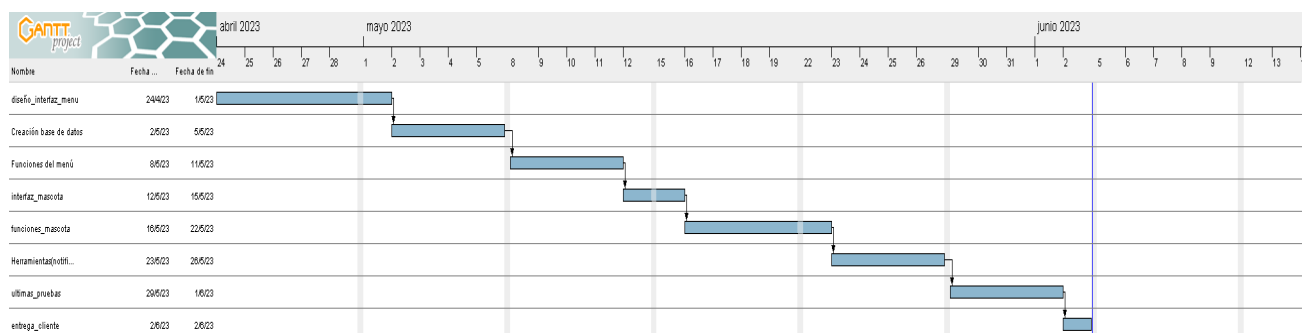
Código 1 Base de datos.....	16
Código 2 Entidad usuario.....	16
Código 3 Entidad de mascota.....	17
Código 4 Entidad historial mascota.....	18
Código 5 Entidad historial limpieza.....	18
Código 6 Consulta mascota.....	19
Código 7 Cargar listas.....	19
Código 8 Actualizar RecyclerView.....	20
Código 9 Menú aplicación.....	21
Código 10 Canal notificación.....	25
Código 11 Creación de alarmManager.....	25
Código 12 Creación de notificación.....	26

- Justificación -

He hecho una aplicación de mantenimiento de mascotas porque me atrae todo lo que es el trato a las mascotas tanto jugar con ellas, prestar atención si tienen algún problema de salud, y sobre todo que no se aburran. Además, he visto que había pocas aplicaciones de móvil de este tipo, había para la adopción de mascotas, pero no para cuidarlas y tener un historial de seguimiento de su cuidado.

La aplicación debe mostrar los datos e indicar cuales hacen falta mejorar para la mascota, tendrá un apartado para mejorar la salud de la mascota dependiendo de los objetivos del usuario los resultados los mostrará de manera que el usuario sepa por dónde empezar a mejorar a su mascota. Por otra parte, la debe tener un historial de la limpieza para tener en cuenta cuánto tiempo lleva sin una higiene, la limpieza se podrá especificar, también tiene que tener información extra sobre posibles infecciones o malestares de la mascota y como evitarlos. Y luego otro apartado de Salud dónde se mostrará un historial del veterinario y un resumen del diagnóstico. Por otra parte, debe tener la opción de crear recordatorios para ayudar a la planificación del usuario y acceso a lugares cercanos de veterinarios y lugares dónde se puede llevarlos.

- Gestión del proyecto -



El proyecto empezaría el 24 de abril, donde se comenzaría por crear los primeros prototipos de la aplicación, hasta el 2 de mayo. Durante el primer bloque de tarea se creará y se establecerá la interfaz principal de la aplicación. El segundo bloque que es la base de datos se organizará como gestionar los datos y las relaciones entre las tablas, además se tendrá que crear las restricciones necesarias para que no se produzca fallo al acceder a las tablas o en la relación entre ellas cuando haya que consultar por ejemplo las mascotas de un usuario. En el tercer bloque se implantarán las funciones del menú, no se tendrá acceso a modificar una mascota, se centra en organizar las mascotas ventanas importantes y añadir mascotas, también se tendrán

apartados para modificar el perfil y añadir recordatorios. Luego la interfaz mascota, contendrá las principales funciones de esta aplicación y la información de cada mascota, las funciones de la interfaz mascota tendrán que ser capaces de actualizar los datos de la mascota si se ha modificado o se ha borrado y actualizar la interfaz con los nuevos datos. El siguiente paso por lo tanto son las funciones de esta interfaz, en este apartado se necesita más tiempo ya que es el más importante y se tiene que implementar un código capaz de soportar cualquier error, se deben controlar las restricciones al modificar una mascota y que los datos sean coherentes entre ellos, además se debe poder acceder a los historiales de la mascota. El siguiente bloque corresponde a las funciones de crear notificaciones, se debe crear un canal de aplicación y un controlador de estos, un gestor que muestre el día que el usuario indico que se muestre, es una funcionalidad que complementa la función principal de la aplicación por lo tanto se debe tener en cuenta después de lo principal, aunque son importantes para una mejor experiencia para el usuario. Por último, se hacen las últimas definitivas buscando los fallos en cualquier lugar de la aplicación o en caculos importantes que hace la aplicación para comprobar si se ha hecho correctamente a prueba de errores. Al final se le entrega la aplicación y su guía al cliente.

- Herramientas utilizadas -

Como herramienta principal se ha utilizado el lenguaje de programación Kotlin para todas las funcionalidades de la aplicación, luego para la interfaz se ha utilizado XML. Con estos dos lenguajes de programación se ha hecho gran parte de la vista y el control de la aplicación. En cuanto a la base de datos se ha utilizado SQLite con la librería de base de datos que proporciona Android, una capa intermedia entre los datos y la aplicación, que es Room, se encarga de toda la parte de la conexión, consultas que se han hecho a la base de datos se han realizado con los métodos de Kotlin y la implementación de Room. Estas herramientas las he utilizado por lo siguiente:

- Kotlin:

Es un lenguaje fácil de entender si lo comparamos con Java requiere de menos código para las mismas tareas, se pueden utilizar las bibliotecas y herramientas de Java. Es un lenguaje oficialmente respaldado para el desarrollo de aplicaciones móviles en Andriod.

- Android Studio:

Es la herramienta oficial para aplicaciones Android y está respaldada por Google. Se tiene una integración completa con el sistema Android, lo cual se podrán utilizar todas las herramientas, SDK y bibliotecas de Android.

- Room:

Proporciona una capa de abstracción sobre SQLite lo que permite interactuar con la base de datos mediante objetos. Proporciona anotaciones que permiten definir la estructura de la base de datos. Realiza comprobaciones en tiempo de compilación para garantizar las operaciones que se han realizado a la base de datos.

- GitHub:

Es una herramienta que se proporciona un buen seguimiento de los cambios que se han hecho en el código. Permite un gran control sobre la aplicación se pueden revertir o deshacer los cambios si es necesario. Se pueden clonar repositorios locales fácilmente para continuar con tu proyecto por dónde lo habías dejado en otro equipo.

- Draw.io:

Esta herramienta se ha utilizado para la creación de los diagramas de la aplicación. Es una herramienta online no se necesita descargar, además es fácil de usar y gratuita. Se pueden crear diagramas de diferentes objetivos del proyecto.

- GanttProject:

Es una aplicación que permite gestionar las fechas del proyecto y llevar a cabo una mejor planificación para las partes del proyecto. Tiene un gran número de opciones para modificar las tareas del proyecto.

- ROOM -

<https://developer.android.com/jetpack/androidx/releases/room?hl=es-419>

- ANDROID STUDIO-

<https://developer.android.com/studio>

-GitHub-

<https://github.com/>

-Draw.io-

<https://app.diagrams.net/>

Para gestión del proyecto:

-GanttProject-

<https://www.ganttproject.biz/>

- Descripción del proyecto -

Análisis:

La aplicación servirá para llevar un control de la salud y la limpieza de las mascotas que tenga el usuario. Se podrá registrar con un nombre y contraseña, luego en el menú de inicio podrá añadir las mascotas que tenga ofreciendo los datos importantes para poder una mejor precisión a la hora de saber las necesidades de la mascota. Cuando seleccione una mascota del menú se abrirá un nuevo menú para la mascota en el que podrá realizar modificaciones en los datos de la mascota o borrarla de la base de datos y ver los datos que tiene y su estado. Tendrá la opción de mejorar la salud de la mascota dependiendo de si el objetivo es buscar una combinación de actividad y reposo para la mascota, o mejorar su actividad, o su alimentación, se recogerán los datos de la mascota y se guiará al usuario por un camino para realizarlo, o si ve necesario otra opción recomendárselo al usuario. Otra función que tendrá es la de poder seguir un historial de las visitas que es hagan al veterinario, se indicará un motivo, la fecha y un resumen del diagnóstico. Por otra parte, también se podrá realizar un seguimiento de la limpieza, mostrando lo que se ha limpiado y la fecha. Por último, se pueden desde el menú de inicio crear recordatorios para una mejor planificación para el usuario.

Por lo tanto, se deben controlar los siguientes posibles problemas, que en el inicio de sesión se controle el usuario y contraseña sean correctos, si se equivoca en uno la aplicación debe informar sobre el error de inicio de sesión. En la pantalla de registro se debe controlar que no haya más de un usuario con el mismo nombre que otro, porque el almacenamiento va a ser local y no tienen que haber más de dos usuarios con los mismos datos. Para las mascotas debe haber un control sobre los datos introducidos los datos deben de ser lógicos entre sí para un mayor control y mejor seguimiento de los datos de la mascota. Si se crea más de una notificación se debe

controlar los identificadores de estas para que el sistema Android no sobrescriba la notificación anterior. Cada vez que se modifique un usuario debe haber un control sobre los nuevos datos introducidos y que estos se guarden correctamente en la base de datos. En modificar mascota debe haber un mismo control sobre los datos para que haya lógica entre los nuevos datos. Los historiales deben corresponder correctamente a una mascota. Por último, en el plan de mejora debe obtener los datos de la mascota para crear consejos más eficaces dependiendo del estado actual de la mascota.

Diseño:

Entidad – Relación:

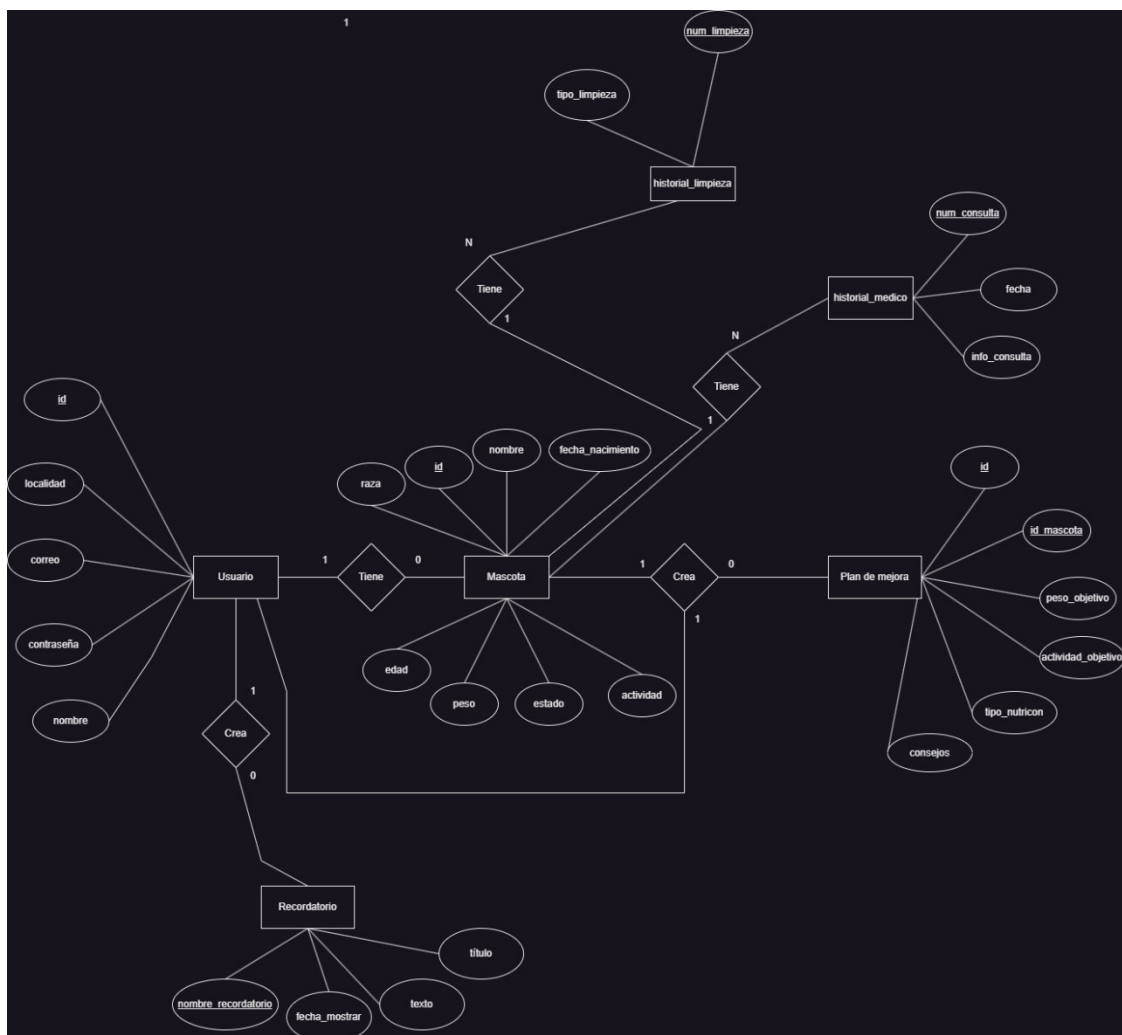


Diagrama 1 Diagrama E/R

Se necesita un Usuario para poder utilizar la app, en esta entidad se guardarán datos básicos, la aplicación va a necesitar el nombre y la contraseña. Luego en la entidad Mascota se necesita todo lo necesario para que se pueda seguir un buen seguimiento de la mascota, el peso, la edad, el tamaño son algunos de los datos necesarios para tener mejor referencia de la mascota y obtener resultados más exactos. Luego como se dijo van a ver dos historiales para tener un seguimiento de la mascota del veterinario o de la limpieza en estas entidades se necesita el identificador de la mascota y luego la información sobre cada historial. Luego la notificación o recordatorios que generará el usuario para tener una buena planificación y luego el sistema Android mediante los métodos para la gestión de notificaciones se encarga de mostrar la notificación en la fecha indicada, por último, en el plan de mejora se necesita una entidad mascota para observar los datos y decidir que recomendaciones dar al usuario para mejorar o la actividad o salud de la mascota. Con cada cambio que tenga la mascota se podrá reflejar en la aplicación por lo tanto los consejos también cambiarán.

Diagrama de clases:

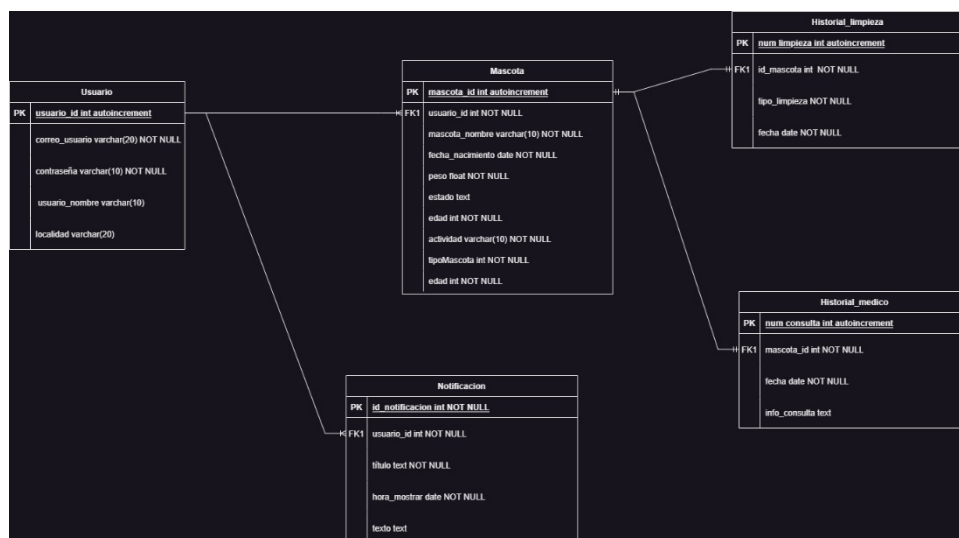


Diagrama 2 Diagrama de Clases

En el paso de tablas se necesitará un id por cada entidad, en la entidad Usuario un usuario puede tener varias mascotas, luego notificaciones simplemente se necesita el id de la notificación luego esta notificación se añade a un canal de notificaciones para gestionarlas y enviarlas en su momento. En la tabla mascota la contendrá un Usuario, para poder obtener el historial de veterinario y limpieza se necesita el identificador de

la mascota correspondiente y estos historiales solamente deben de ser de una mascota, con la información importante de cada una. Por último, el plan de mejora que es el responsable de calcular la mejor opción para mejorar a la mascota se necesitará la información de la entidad mascota, cada vez que se actualice a la mascota las opciones de mejora cambiarán.

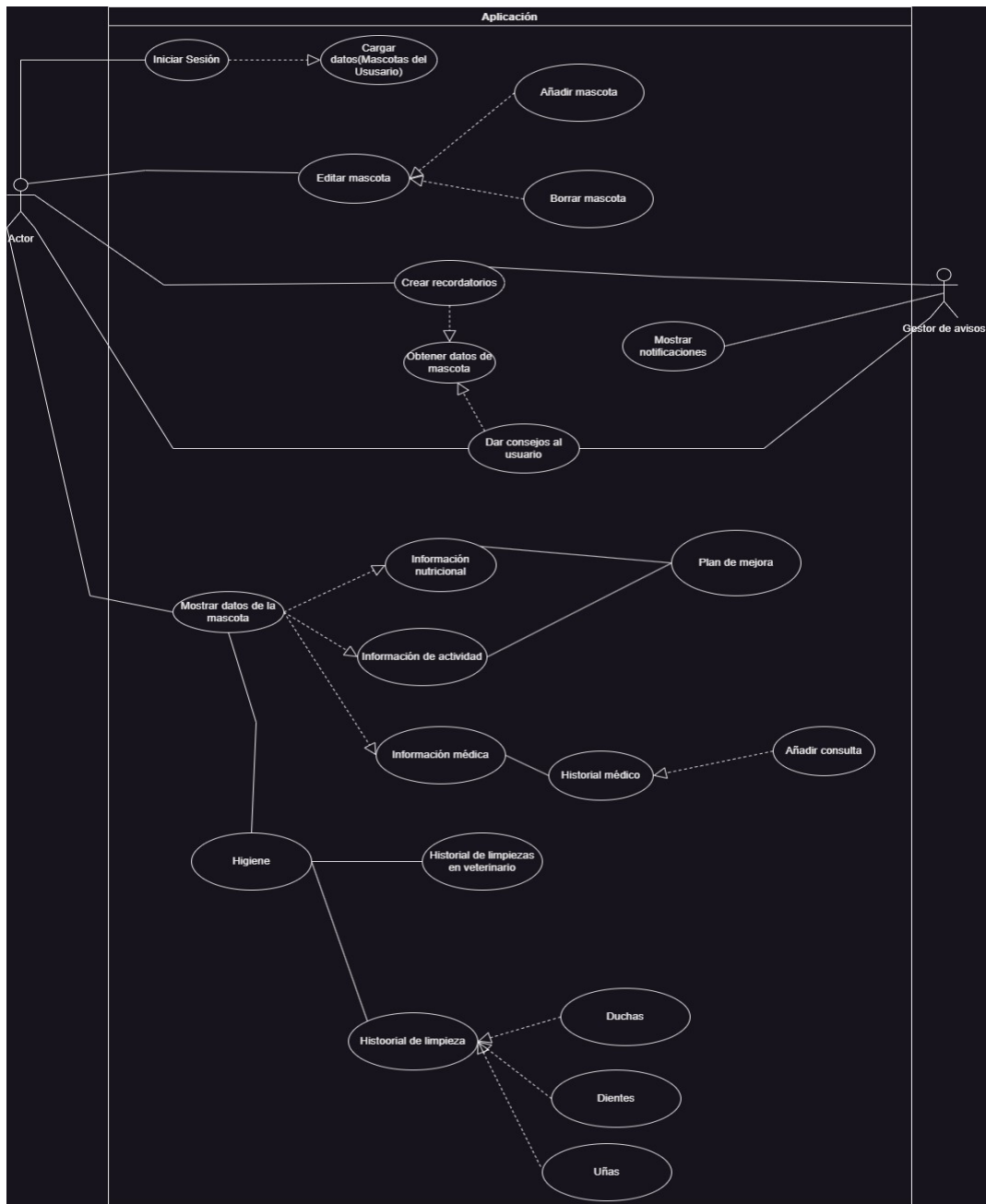


Diagrama 3 Casos de uso

En este diagrama de casos de uso se puede ver cómo el usuario debe iniciar sesión para cargar los datos que tenga en la aplicación, luego podrá añadir o borrar mascota, también la podrá modificar al elegir cuál mascota quiere mejorar o ver sus datos. Dentro de esta mascota puede acceder a los historiales de mascota. Podrá también crear los recordatorios que quiera para planificar mejor su tiempo. Luego nuevamente dentro de una mascota se puede acceder a la mejora, en esta mejora puede ser de Actividad, Salud, Peso, etc. Este método recogerá los datos de la mascota y generará los consejos necesarios al usuario.

- Iniciar Sesión:

En este caso se utiliza la tabla Usuario para buscar e identificar al Usuario que se ha introducido en los campos Nombre y Contraseña. Con esta acción se extiende la siguiente automáticamente. En la siguiente se cargarán todos los datos que contenga este Usuario en el programa.

- Editar mascota:

Se utilizan las tablas Mascota y Usuario. Cuando se añade a una mascota se le vincula el identificador del Usuario que anteriormente inició sesión en la aplicación. Si se quiere eliminar se eliminarán todos los datos de esta mascota incluido los historiales que tenga.

- Crear recordatorio:

En este caso se utilizan los campos que haya proporcionado el Usuario para que se muestren en el sistema, en esta acción se crea automáticamente un identificador para la notificación que a su vez se añade a un canal de notificaciones de la aplicación para que el sistema Android las gestione y las muestre en la fecha que el Usuario indico.

- Dar consejos al usuario:

Se mostrarán los recordatorios que el Usuario creó. Luego, además se encarga de gestionar y mostrar los consejos del plan de mejora dependiendo de los datos de la mascota.

- Mostrar datos de la mascota:

Se mostrarán tanto los datos básicos como los historiales que tenga la mascota. Esta acción corresponde al cargar el menú de la mascota, la tabla principal es Mascota, y luego para saber a qué Usuario corresponde esta mascota se le debe pasar el identificador del Usuario mediante una clave foránea.

- Plan de mejora:

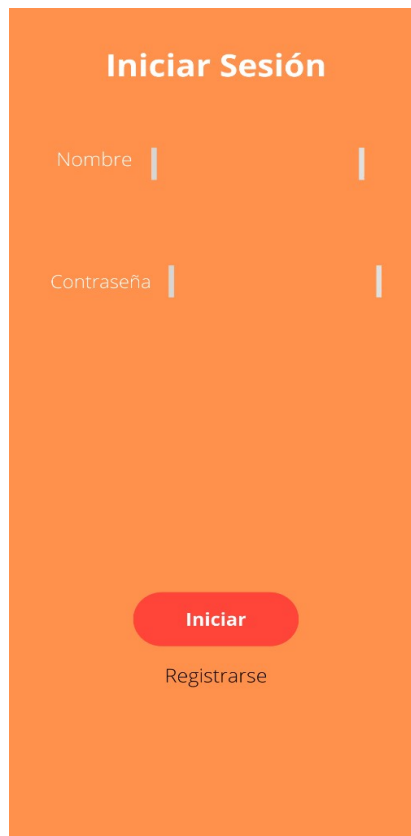
Se recogen los datos de la mascota que se cargó, para tener una mayor eficacia en el plan de mejora se recogen todos los datos, además depende del tipo de consejos que haya elegido el usuario recibirá diferentes consejos. Por lo tanto, se utiliza la tabla Mascota para analizar sus datos.

- Historial de limpieza:

En el historial de limpieza el Usuario podrá añadir el tipo de limpieza que haya realizado a la mascota, también podrá especificar la fecha para llevar un seguimiento. Se utilizará la tabla de HistorialLimpieza y la de Mascota para que este historial solamente aparezca en la mascota indicada.

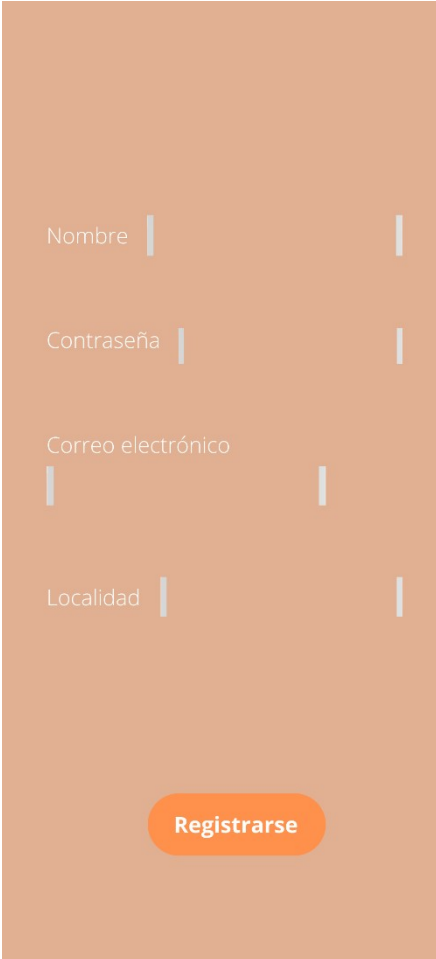
Con estos diagramas se puede obtener los datos necesarios para desarrollar la aplicación se divide en Usuario que puede modificar, añadir o borrar mascotas, y luego cada mascota dependiendo de los datos que tenga se podrá obtener datos o consejos diferentes en el plan de mejora, además cada mascota tendrá su historial para tener un mejor seguimiento de esta.

Prototipo interfaz:

El prototipo de interfaz de inicio de sesión tiene un fondo naranja sólido. En la parte superior, el título "Iniciar Sesión" está en blanco. Debajo, hay dos campos de entrada con el texto "Nombre" y "Contraseña" en gris. Cada campo tiene una barra vertical blanca a la izquierda y una barra vertical blanca a la derecha. En la parte inferior, hay un botón redondeado rojo con el texto "Iniciar" en blanco, y debajo de él, el texto "Registrarse" en gris.

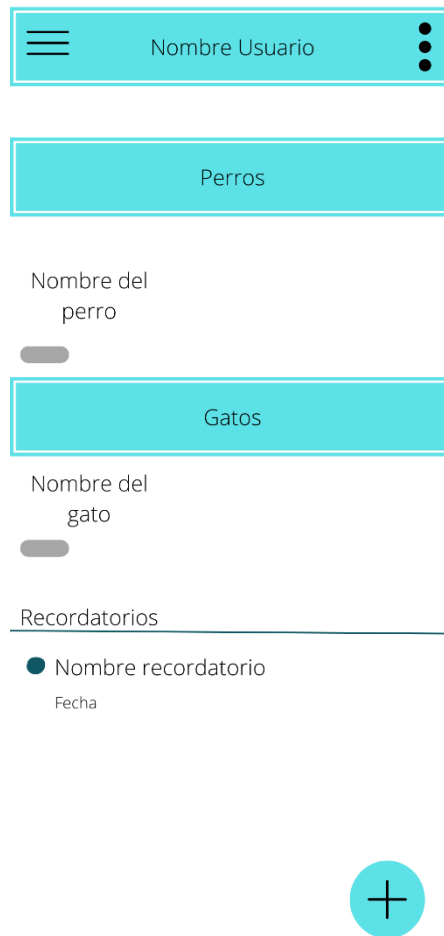
Prototipo 1

En esta imagen se puede observar cuál va a ser la primera pantalla que se podrá ver de la aplicación. Contiene los dos campos necesarios para el inicio de sesión Nombre y Contraseña, luego dos botones para añadir un Usuario si no está registrado en la base de datos, o inicio de sesión.

A vertical rectangular screen with a solid light orange background. It features four input fields stacked vertically, each with a light gray label on the left and a vertical line on the right indicating the input area. The labels are 'Nombre', 'Contraseña', 'Correo electrónico', and 'Localidad'. Below these fields is a single orange rounded rectangular button with the white text 'Registrarse' centered on it.

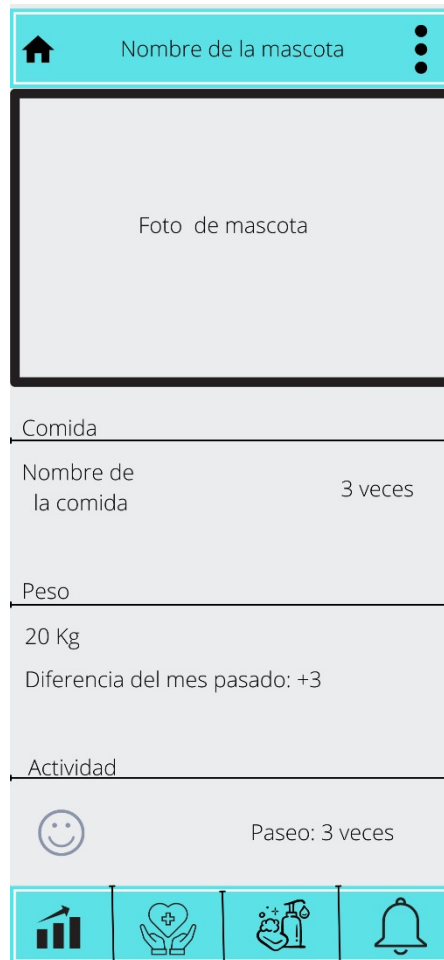
Prototipo 2

Para la pantalla de registro se recogen datos básicos del Usuario, pero los que debe rellenar obligatoriamente són Nombre y Contraseña para poder utilizar la aplicación.



Prototipo 3

Ésta sería la pantalla de inicio de la aplicación se divide en dos listas dinámicas por cada tipo de animal. Se utilizará dos RecyclerView para poder personalizar el cómo se va a ver cada mascota añadida Gato o Perro, también servirá para que cada vez que haya una actualización de la lista de mascotas del Usuario se encargue de modificar su aspecto, y tendrá un escuchador que se encargará de devolver la posición de la mascota de la lista y así poder abrir el menú correspondiente de esa mascota. Luego se ve que se muestran los recordatorios, aunque al final se creó una ventana de recordatorios a parte en el menú. Por último, en esta ventana también se puede ver el botón para añadir mascota se abriría una ventana para elegir si añadir gato o perro y luego completar los datos de la mascota.



Prototipo 4

Este sería el menú que aparece al pulsar una mascota de la lista anteriormente mencionada, se diseñó de manera que se pudieran ver los datos básicos de la mascota al principio, abajo se puede ver el menú y se puede identificar el plan de mejora, el historial de veterinario, el historial de limpieza y luego el de notificaciones que cómo se dijo antes al final paso a ser parte del menú de inicio. El menú de opciones de arriba a la derecha contendrá las opciones de borrar mascota y modificar la mascota.

Desarrollo:

El proyecto se ha separado de manera que el primer paso fue la creación de la base de datos con las entidades necesarias, gracias a la biblioteca de Room que proporciona Android podremos crear esta base de datos directamente desde Android Studio.

Base de Datos:

```
@Database(entities = arrayOf(UsuarioEntity::class, MascotaEntity::class, HistorialMedicoEntity::class, HistorialLimpiezaEntity::class), version = 1, exportSchema = false)
abstract class appDatabase : RoomDatabase() {
    # abstract fun usuarioDao() : UsuarioDAO
    # abstract fun mascotaDao() : MascotaDAO
    # abstract fun historialMedicoDao() : HistorialMedicoDAO
    # abstract fun historialLimpiezaDao() : HistorialLimpiezaDAO
    companion object {
        @Volatile
        private var INSTANCE: appDatabase? = null
        fun getDatabase(context: Context): appDatabase {
            return INSTANCE?.synchronized(lock: this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    appDatabase::class.java,
                    name: "fidofriend"
                )
                    .fallbackToDestructiveMigration()
                    .build()
                INSTANCE = instance
                return instance
            }
        }
    }
}
```

Código 1 Base de datos

Esta es la clase que contiene la base de datos. Se especifican las entidades que se van a utilizar y luego los métodos abstractos DAO de cada entidad. Al instanciar la clase se revisará si se tiene una instancia de la base de datos, con `@Volatile` los hilos recogerán la última versión de la instancia para garantizar la consistencia de los datos y devolverá esa misma instancia. Si no existe una instancia de la base de datos se creará con el método `databaseBuilder()`.

Entidades:

- Usuario:

```

9      @Entity(tableName = "usuario",
10             indices = arrayOf(
11                 Index(
12                     value = arrayOf("nombre","email"),
13                     name = "idx_nombre_email",
14                     unique = true
15                 )
16             )
17     )
18     data class UsuarioEntity(
19         @PrimaryKey(autoGenerate = true)
20         @ColumnInfo(name = "usuario_id")
21         var id: Int,
22         @ColumnInfo(name = "nombre")
23         var nombre: String,
24         @ColumnInfo(name = "contrasena")
25         var contrasena: String,
26         @ColumnInfo(name = "email")
27         var email: String,
28         @ColumnInfo(name = "localidad")
29         var localidad: String
30     )

```

Código 2 Entidad usuario

En cuanto a las entidades que representan las tablas de la base de datos, en Usuario se puede observar la clave primaria que es el identificador tipo Int, además arriba hay un índice que indica que nombre y el correo tienen que ser únicos, es decir dentro del almacenamiento local de la aplicación en la tabla Usuario no pueden haber más de dos usuarios con el mismo nombre o correo.

- Mascota:

```

10 @Entity(tableName = "mascota",
11         foreignKeys = [ForeignKey(
12             entity = UsuarioEntity::class,
13             childColumns = ["usuario_id"],
14             parentColumns = ["usuario_id"],
15             onDelete = CASCADE
16         )],
17         indices = arrayOf(
18             Index(
19                 value = arrayOf("usuario_id"),
20                 name = "idx_mascota_usuario"
21             )
22         )
23     )
24
25 @data class MascotaEntity(
26     @PrimaryKey(autoGenerate = true)
27     @ColumnInfo(name = "mascota_id")
28     var id_mascota: Int,
29     @ColumnInfo(name = "mascota_nombre")
30     var nombre: String,
31     @ColumnInfo(name = "fecha_nacimiento")
32     var fecha_nacimiento: String,
33     @ColumnInfo(name = "peso")
34     var peso: Float,
35     @ColumnInfo(name = "estado")
36     var estado: String,
37     @ColumnInfo(name = "edad")
38     var edad: Int,
39     @ColumnInfo(name = "actividad")
40     var actividad: String,
41     @ColumnInfo(name = "porte")
42     var porte: String,
43     @ColumnInfo(name = "perroGato")
44     var perroGato: Int,
45     @ColumnInfo(name = "usuario_id")
46     var usuario_id: Int
47 )

```

Código 3 Entidad de mascota

En la entidad Mascota tiene un identificador como Usuario además tiene una columna en la que se guarda el usuario que es dueño de esta mascota. Por lo tanto, arriba en los índices tenemos una clave foránea que hace referencia al identificador de Usuario y si se elimina un usuario se borrarán todas las mascotas del mismo.

- Historial veterinario:

```

10  @Entity(tableName = "historial_mascota",
11      foreignKeys = [ForeignKey(
12          entity = MascotaEntity::class,
13          childColumns = ["mascota"],
14          parentColumns = ["mascota_id"],
15          onDelete = CASCADE
16      )],
17      indices = arrayOf(
18          Index(
19              value = arrayOf("mascota"),
20              name = "idx_historial_mascota_mascota"
21          )
22      ))
23  data class HistorialMedicoEntity(
24      @PrimaryKey(autoGenerate = true)
25      @ColumnInfo(name = "historial_mascota_id")
26      var historialId: Int,
27      @ColumnInfo(name = "titulo")
28      var titulo: String,
29      @ColumnInfo(name = "fecha")
30      var fecha: String,
31      @ColumnInfo(name = "descripcion")
32      var descripcion: String,
33      @ColumnInfo(name = "mascota")
34      var mascotaId: Int
35  )
36

```

Código 4 Entidad historial mascota

En esta entidad se necesita la mascota a la que está asociada, así el historial se controla que no aparezcan datos de otra mascota, esto se hace con el uso de la clave foránea del identificador de mascota para identificar a que mascota se le debe añadir el historial. Y por último indicar que si se borra una mascota se debe borrar además este historial. El historial de limpieza viene a ser lo mismo, pero para guardar otros datos como las fechas de la limpieza y el tipo de limpieza.

```

10  @Entity(tableName = "historial_limpieza",
11      foreignKeys = [ForeignKey(
12          entity = MascotaEntity::class,
13          childColumns = ["mascota"],
14          parentColumns = ["mascota_id"],
15          onDelete = CASCADE
16      )],
17      indices = arrayOf(
18          Index(
19              value = arrayOf("mascota"),
20              name = "idx_historial_limpieza_mascota"
21          )
22      ))
23  data class HistorialLimpiezaEntity(
24      @PrimaryKey(autoGenerate = true)
25      @ColumnInfo(name = "historial_limpieza_id")
26      var historialId: Int,
27      @ColumnInfo(name = "parte_limpieza")
28      var parteLimpieza: String,
29      @ColumnInfo(name = "fecha")
30      var fecha: String,
31      @ColumnInfo(name = "mascota")
32      var mascotaId: Int
33  )
34

```

Código 5 Entidad historial limpieza

Interfaz menú:

El siguiente paso fue el diseño que tendría el menú apenas se iniciaba sesión. Como se vio anteriormente en el prototipo el menú de inicio se dividirá en la ventana principal con las dos listas de Perro y Gato, luego una ventana para modificar el perfil, y por último otra para crear recordatorios. Luego también cuando se pulsa el botón de añadir mascota aparecerá otra ventana para ir rellenando datos de la mascota que se va a añadir. Cuando se añada una mascota las dos listas se recargarán para ver si hay nuevas mascotas, la manera en la que se van a rellenar estas listas es mediante llamadas a la base de datos al DAO de la tabla mascota, se hará mediante corrutinas en hilos secundarios para no bloquear el hilo principal que es el encargado de la interfaz.

```
29 //Obtener listado de perros/gatos
30 @Query(value = "SELECT * FROM mascota WHERE perroGato LIKE :tipoMascota AND usuario_id = :id")
31 fun getPerroGato(tipoMascota: Int, id: Int): MutableList<MascotaEntity>
32
```

Código 6 Consulta mascota

```
74 private fun cargar(nombre: String) {
75     viewLifecycleOwner.lifecycleScope.launch { this: CoroutineScope
76         withContext(Dispatchers.IO) { this: CoroutineScope
77             usuarioEntity = usuarioRepository.getUsuarioByName(nombre)
78             gatos = mascotaRepository.getPerroGato( tipoMascota: 2, usuarioEntity.id)
79             perros = mascotaRepository.getPerroGato( tipoMascota: 1, usuarioEntity.id)
80         }
81         setUpRecyclerViewGato()
82         setUpRecyclerViewPerro()
83     }
84 }
```

Código 7 Cargar listas

```

86     @SuppressWarnings("NotifyDataSetChanged")
87     private fun setUpRecyclerViewGatos() {
88         if (gatos.isNotEmpty()) {
89             adapterGatos = GatoAdapter(gatos,
90                 ) { MascotaEntity
91                 var intent = Intent(activity, MascotaActivity::class.java).putExtra( name: "idMascota", it.id_mascota)
92                 startActivity(intent)
93             }
94             recyclerView = binding.recyclerViewGatos
95             recyclerView.setHasFixedSize(true)
96             recyclerView.layoutManager = LinearLayoutManager(this.requireContext(), LinearLayoutManager.HORIZONTAL, reverseLayout: false)
97             recyclerView.adapter = adapterGatos
98             adapterGatos.notifyDataSetChanged()
99         }
100     }
101
102 }
103
104 @SuppressWarnings("NotifyDataSetChanged")
105 private fun setUpRecyclerViewPerros() {
106     if (perros.isNotEmpty()) {
107         adapterPerros = PerroAdapter(perros
108             ) { MascotaEntity
109             var intent = Intent(activity, MascotaActivity::class.java).putExtra( name: "idMascota", it.id_mascota)
110             startActivity(intent)
111         }
112         recyclerView = binding.recyclerViewPerros
113         recyclerView.setHasFixedSize(true)
114         recyclerView.layoutManager = LinearLayoutManager(this.requireContext(), LinearLayoutManager.HORIZONTAL, reverseLayout: false)
115         recyclerView.adapter = adapterPerros
116         adapterPerros.notifyDataSetChanged()
117     }
118 }

```

Código 8 Actualizar RecyclerView

En los métodos setUpRecyclerView“ “ () se avisará a los recyclerView que vuelvan a cargar las listas con las nuevas listas del usuario, cuando se añada o se elimine una mascota y se vuelva a cargar el menú de inicio se ejecutará este código. En la ventana perfil habrá un botón para cada campo que se quiera modificar, además abajo habrá un botón de borrar cuenta, y luego te envía directamente a la pantalla inicio de sesión, se eliminarán también las mascotas y sus datos del usuario eliminado. Luego la ventana de notificaciones se indicará la fecha que se quiere recibir la notificación y un Título y texto. Después de diseñar el menú de inicio se podrán hacer las funciones para iniciar la base de datos y empezar a añadir, borrar datos. Para la comunicación entre las ventanas se ha utilizado el NavigationView con el NavigationUI que permite configurar automáticamente la barra de navegación.

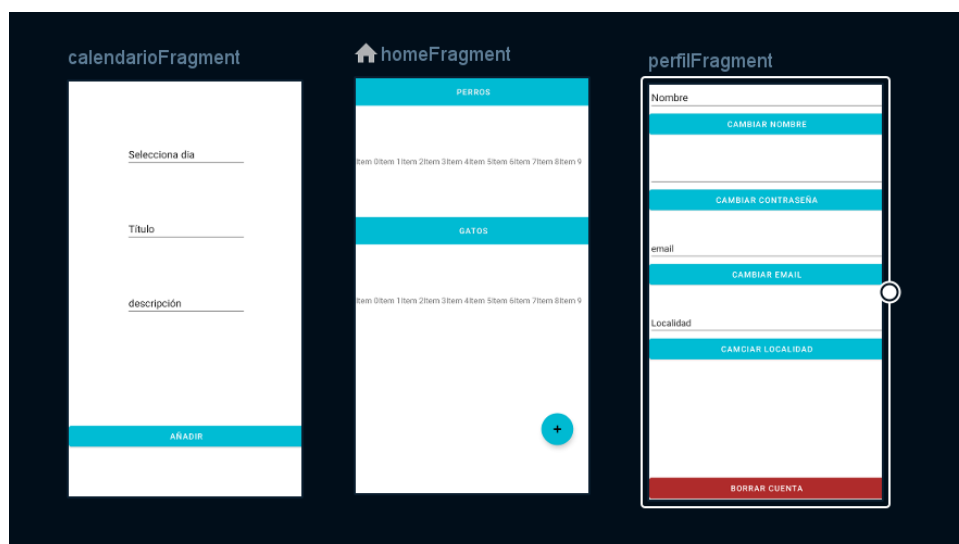
Activity que contiene las ventanas del menú de inicio y las gestiona con NavigationUI:

```

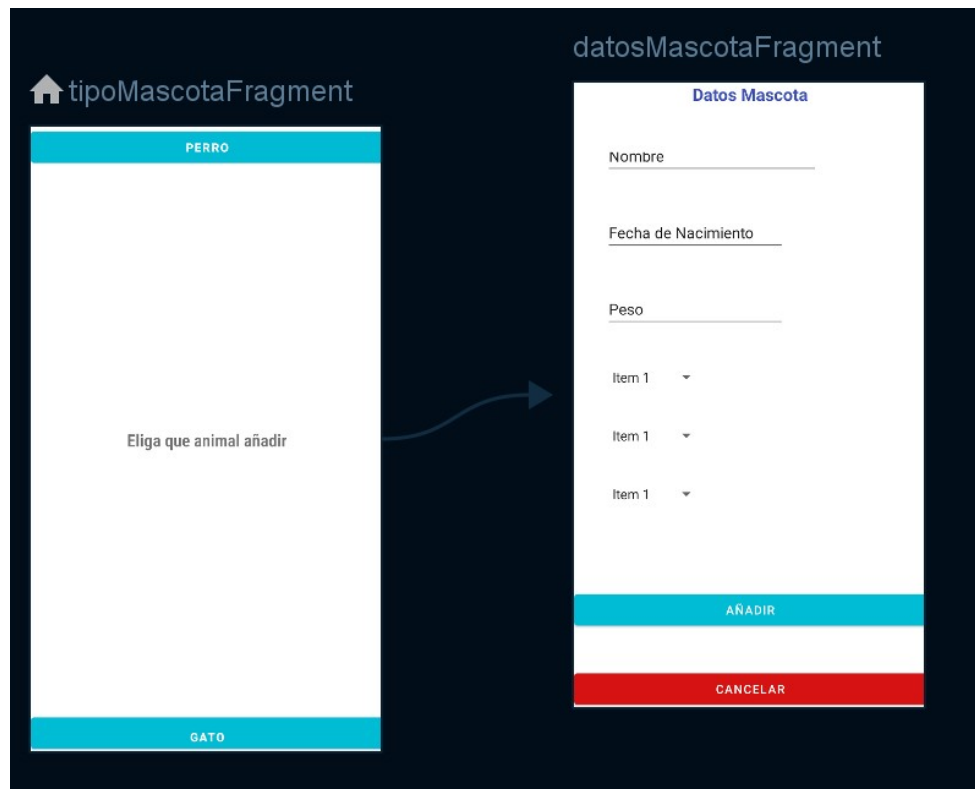
13 class MenuActivity : AppCompatActivity() {
14     private lateinit var binding: ActivityMenuBinding
15     private lateinit var appBarConfiguration: AppBarConfiguration
16     private lateinit var db: appDatabase
17     override fun onCreate(savedInstanceState: Bundle?) {
18         super.onCreate(savedInstanceState)
19         setContentView(ActivityMenuBinding.inflate(layoutInflater).also { binding = it }.root)
20         db = appDatabase.getDatabase(applicationContext)
21         setSupportActionBar(binding.toolbar)
22
23         val navHostFragment = supportFragmentManager.findFragmentById(R.id.nav_host_fragment) as NavHostFragment
24         val navController = navHostFragment.navController
25
26         appBarConfiguration = AppBarConfiguration(setOf(
27             R.id.homeFragment,
28             R.id.perfilFragment
29         ),
30             binding.menuLayout
31         )
32
33         NavigationUI.setupWithNavController(binding.navView, navController)
34         NavigationUI.setupWithNavController(binding.toolbar, navController, appBarConfiguration)
35
36     }
37
38     fun getDB(): appDatabase = db
39
40 }

```

Código 9 Menú aplicación



Grafo 1 Interfaz menú



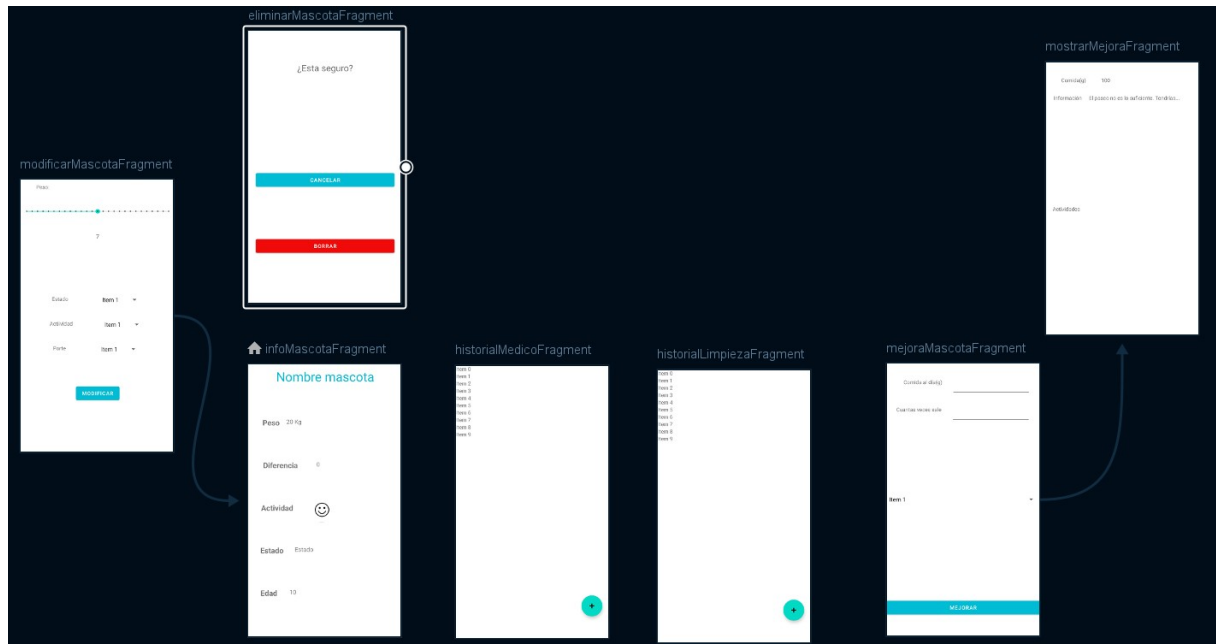
Grafo 2 Interfaz añadir mascota

El menú quedaría como se muestra en los grafos, se pueden observar las flechas que servirán para abrir los fragmentos para rellenar la información de la mascota que se quiere añadir. Esta acción no la gestiona el NavigationUI, por lo tanto, manualmente se ha tenido que especificar que el la Actividad padre de los fragmentos para añadir la mascota reciban la orden de crearse al pulsar el botón de añadir mascota del menú de inicio. Cuando se complete se podrá cancelar o añadir, ambas devuelven al menú de inicio, pero si se pulsó añadir se actualizará la lista de Perro y Gato. Al pulsar una mascota de las listas del menú de inicio se cargará la interfaz de mascota.

Interfaz mascota:

En esta parte del proyecto se necesitará nuevamente realizar un menú y un grafo para los apartados que tenga la mascota. En este caso se utilizará un BottomNavigationView, un menú que muestra los apartados abajo. En este menú aparecerá tanto la información general de la mascota como el peso, la edad, su nombre, y más. Luego se tendrá acceso a los historiales, para añadir y tener un seguimiento de estos, y luego se tendrá también el apartado de mejorar el estado de la mascota mediante consejos. Se vio que para mostrar los consejos se podía especificar

de que tipo para mejorar las respuestas. Además, hacer una tabla en la base de datos no serviría, se necesita una función que reciba la entidad mascota, y con los datos que tenga y el tipo de mejora que elija el usuario podrá dar diferentes consejos. Por otra parte, se ha implementado también otro menú arriba para modificar los datos de la mascota y borrar la mascota.



Grafo 3 Interfaz menú mascota

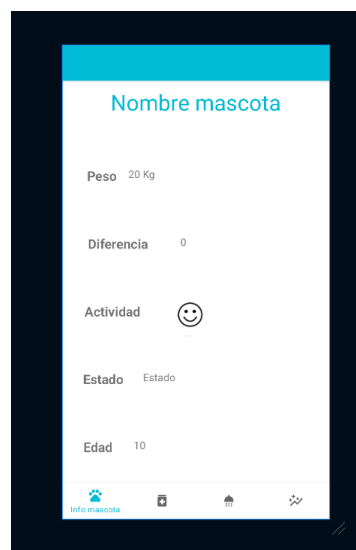


Ilustración 1 Ventana información

Tal como se ve en la imagen se puede ver el diseño que tendría el menú finalmente, y como se dijo anteriormente en el apartado de modificar mascota se han restringido datos que no se podrán cambiar, solamente se podrán modificar datos que vayan cambiando con el tiempo por ejemplo el peso, la actividad que tiene, el tipo de vida que tiene, si está enfermo, etc.

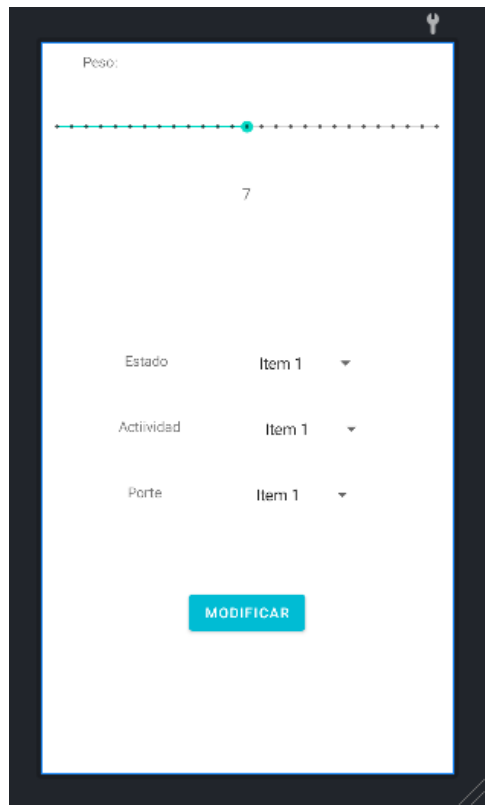


Ilustración 2 Modificar mascota

Estos cambios se verán reflejados en la ventana de información de la mascota y al recibir consejos de mejora de la mascota. Al pulsar la opción borrar mascota del menú de arriba se abrirá una nueva ventana para verificar que el usuario quiere realizar esta acción. Si se cancela la acción devolverá a la ventana de información de la mascota, si se pulsa borrar devolverá al usuario al menú de inicio. Por último, se volvió al menú de inicio para terminar la aplicación implementando las funciones de las notificaciones. Como se dijo anteriormente la ventana de notificaciones tendrá opción para elegir la fecha y otros dos campos para completar el contenido que mostrará la notificación. En este apartado se necesitará crear un canal de notificaciones para la aplicación, este canal servirá para que el sistema identifique a que aplicación pertenece ese canal.

```

100     @RequiresApi(Build.VERSION_CODES.O)
101     @SuppressWarnings("ServiceCast")
102     private fun crearCanal() {
103         if (Build.VERSION.SDK_INT ≥ Build.VERSION_CODES.O) {
104             val channel = NotificationChannel(
105                 MY_CHANNEL_ID,
106                 name: "Canal de FidoFriend",
107                 NotificationManager.IMPORTANCE_DEFAULT
108             ).apply { this: NotificationChannel
109                 description = "Canal de recordatorios"
110             }
111             val notificationManager: NotificationManager =
112                 requireContext().getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
113             notificationManager.createNotificationChannel(channel)
114         }
115     }
116
117     companion object {
118         const val MY_CHANNEL_ID = "mi_canal_id"
119     }
120 }

```

Código 10 Canal notificación

En la siguiente imagen se ve como se crea este canal. Al pulsar el botón añadir un recordatorio la fecha se utilizará para especificar la fecha en la que se mostrará la notificación. Se hará un PendingIntent, que básicamente contiene a la clase que crea las características de la notificación y a su vez abre una parte de la aplicación que se especifica al pulsarla.

```

74     @RequiresApi(Build.VERSION_CODES.O)
75     private fun scheduleNotification() {
76         try {
77             val NOTIFICATION_ID = Random.nextInt()
78             val intent =
79                 Intent(requireContext().applicationContext, AlarmNotificacion::class.java).apply { this: Intent
80                     putExtra( name: "title", binding.edTitulo.text.toString())
81                     putExtra( name: "description", binding.edTextDescrip.text.toString())
82                     putExtra( name: "NOTIFICATION_ID", NOTIFICATION_ID)
83                 }
84             val pendingIntent = PendingIntent.getBroadcast(
85                 requireContext().applicationContext,
86                 NOTIFICATION_ID,
87                 intent,
88                 flags: PendingIntent.FLAG_IMMUTABLE or PendingIntent.FLAG_UPDATE_CURRENT
89             )
90
91             val notoficationFecha = fecha!!.atStartOfDay().toInstant(ZoneOffset.UTC).toEpochMilli()
92             val alarmManager =
93                 requireContext().getSystemService(Context.ALARM_SERVICE) as AlarmManager
94             alarmManager.setExact(AlarmManager.RTC_WAKEUP, notoficationFecha, pendingIntent)
95         } catch (e: Exception) {
96             Toast.makeText(requireContext(), text: "Elija una fecha válida y rellene todos los campos", Toast.LENGTH_SHORT).show()
97         }
98     }

```

Código 11 Creación de alarmManager

```

25     private fun crearNotificacion(
26         context: Context,
27         titulo: String,
28         mensaje: String,
29         NOTIFICATION_ID: Int?
30     ) {
31
32         val intent = Intent(context, LoginActivity::class.java).apply { this: Intent
33             flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
34         }
35
36         val flag = if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) PendingIntent.FLAG_IMMUTABLE else 0
37         val pendingIntent = PendingIntent.getActivity(context, requestCode = 0, intent, flag)
38
39         val notification = NotificationCompat.Builder(context,
40             RecordatoriosFragment.MY_CHANNEL_ID
41         )
42             .setSmallIcon(R.drawable.sym_def_app_icon)
43             .setContentTitle(titulo)
44             .setContentText(mensaje)
45             .setContentIntent(pendingIntent)
46             .setPriority(NotificationCompat.PRIORITY_DEFAULT)
47             .setAutoCancel(true)
48             .build()
49
50         val manager = context.getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
51         manager.notify(NOTIFICATION_ID!!, notification)
52     }
53

```

Código 12 Creación de notificación

Luego se puede ver que se realiza una instancia de AlarmManager, esta clase es la encargada de avisar al sistema Android sobre esta nueva notificación. Luego se inicializa una clase que es la que diseña como se va a ver la notificación, en este caso se pasan el título y la descripción que proporcionó el usuario. Y así es como se gestionó finalmente las notificaciones, la parte de mostrar la notificación sin que la aplicación se esté ejecutando ya se encarga el sistema Android.

Pruebas:

Al inicio se tuvo que elegir entre una conexión a MySQL o SQLite, probando diferentes maneras de conexión a la base de datos surgieron dificultades, luego se buscó información sobre crear una base de datos local, en este caso ya se había utilizado una herramienta en el curso que era Room. Como se dijo Room proporcionaba una capa de abstracción entre la base de datos y el código, el resultado

fue una mayor eficacia a la hora de conectar la base de datos la aplicación, además se entendía mucho más la documentación que había sobre esta herramienta, entonces finalmente se eligió esta herramienta de base de datos.

Durante el desarrollo surgieron diferentes métodos para la consistencia de los datos, por un lado, la creación de la clase viewmodel que utiliza una actualización en los LiveData() cada vez que se guarda el resultado de una consulta sin bloquear el hilo principal, este LiveData() persiste en todo momento de la aplicación hasta que se sobrescriba o se cierre la aplicación, la otra opción era realizar las consultas mediante un repositorio en el que contenga los métodos de los DAO y se llame mediante hilos secundarios. Se probó primero con la opción recomendada que era utilizar una clase viewmodel encargada de gestionar los resultados de las consultas, pero se encontró una dificultad al compartir la clase viewmodel entre los fragmentos de un menú. Se buscaron alternativas, pero los datos no se guardaban correctamente en el viewmodel. Finalmente, se cambió la manera en la que se llamaba al DAO de cada entidad, se utilizó la opción de los repositorios. De todas formas, se mientras se implementaba la opción del viewmodel se pudo obtener más información y conocimiento sobre esta clase.

Documentación:

Los siguientes documentos forman parte del proyecto:

- Memoria formato PDF: "IES SERPIS. 2022-23. Proyecto DAM. KevinRodrigoBuenañoEvans.pdf".
- Instrucciones de instalación de FidoFriend.
- Archivo comprimido en zip del proyecto.
- Instalador entorno de desarrollo Android Studio.

Estos archivos son los que forman parte del soporte digital, para ver todos los archivos que se van a entregar.

Trabajos futuros:

Los siguientes puntos son las funcionalidades que se podrán implementar para mejorar el rendimiento y tener una aplicación más completa:

- Auto inicio de sesión: En muchas aplicaciones de Android cuando inicias sesión al cerrar y volver a abrir tu sesión se inicia automáticamente, esto es gracias a unas de las funcionalidades que nos proporciona la biblioteca de Android, es SharedPreferences.
- Apartado configuración: Este es otro aspecto al que se le puede implementar, tener unas opciones para que el usuario pueda configurar sus propias preferencias, como el tema de la aplicación, o ciertas funciones sean más fáciles y viables de acceder en este apartado.
- Confirmación por correo electrónico: Para asegurarse de que las cuentas no sean falsas se podrá implementar una confirmación por correo electrónico para asegurar la autenticidad de las cuentas.
- Autorizaciones de datos personales: Una importante mejora es la de avisar y pedir al usuario sobre el almacenamiento de datos personales tales como el nombre el correo electrónico y la ubicación, así se garantiza la privacidad de los usuarios y la regulación de protección de datos.
- Alimentos: Una de las ampliaciones que más se han aconsejado es la de un apartado en la que se muestre una lista de alimentos que la mascota puede comer, cantidades, y a que edades es mejor darles ese alimento.
- Dividir por razas: Otra ampliación de la aplicación fue el de dividir los perros y gatos por razas, por lo tanto, tendrían diferentes cuidados dependiendo del tipo de raza.
- Enfermedades: Una funcionalidad nueva que se ha pensado también es la de implementar un apartado de salud y enfermedades que pueden afectar a la mascota, y tratar de dar consejos e información que orienten y ayuden al dueño a revisar y cuidar de su mascota.
- Lugares cercanos: Por último, la funcionalidad que permita realizar peticiones a Google Maps para encontrar lugares importantes cercanos como veterinarios, tiendas de comida de mascotas, o lugares interesantes a los que ir a pasear con la mascota. Con esta funcionalidad se podrá hacer uso de la ubicación, campo opcional que se pide al crear una cuenta.

Conclusiones:

En conclusión, este proyecto me ha dado la oportunidad de aplicar los conocimientos aprendidos del curso, y además seguir aprendiendo nuevas funciones y maneras de organizar un proyecto, ha habido puntos en los que no avanzaba por las dificultades que un proyecto puede dar, aun así, no he parado de buscar e informarme para seguir adquiriendo conocimiento y aplicar otros caminos para seguir con el proyecto. El objetivo principal de mi proyecto desde un principio ha sido poder seguir un cuidado de las mascotas, poder mejorar su salud, dar consejos al usuario para que tenga opciones y diferentes maneras de cuidar a la mascota dependiendo de su estado. Para detectar y resolver los errores o mejorar la calidad en general de la aplicación se ha utilizado un emulador de Android Studio, Pixel 4 de Google.

Las partes a destacar del proyecto son en la elección de que herramienta de base de datos utilizar, entre MySQL y SQLite. La importancia que más le di fue el tema de la documentación, la propia página de desarrollo de Android recomendaba utilizar el almacenamiento local con SQLite, además proporcionaban una documentación que era fácil de entender. Por otra parte, información sobre la conexión de MySQL en Android había poca por lo que sumado a que durante el curso se vio la utilidad que daba la librería de Room con una capa de abstracción entre el código y SQLite, me decante por utilizar Room. Toda la documentación de base de datos local de la página de desarrollo de Android se utilizaba Room. Durante la realización de la aplicación me encontré con otro punto importante en el proyecto que fue elegir entre utilizar la clase viewmodel, o utilizar corrutinas al llamar a los métodos de los DAO de las entidades. En este punto realice bastantes pruebas con viewmodel, me informe también en internet, pero surgieron dificultades en la persistencia de datos un punto muy importante en la realización de la aplicación. Por lo que opté por el otro camino finalmente, también realicé pruebas para comprobar si los datos se recibían correctamente, hice varias depuraciones a lo largo del proyecto para comprobar que no hubiera un fallo al consultar algún dato.

En definitiva, la realización de esta aplicación me ha llevado ver los conocimientos ganados durante el curso, también me ayudó en la realización del proyecto a juntar cada uno de estos conocimientos y así poder crear una aplicación con la función de ayudar a orientar a los dueños de las mascotas para tener un seguimiento sobre la salud de la mascota, además dar consejos dependiendo del estado que tenga y la opción darle una opción de recordatorios mediante notificaciones para que el dueño tenga una mejor planificación. Seguiré aprendiendo y mejorando de la aplicación aplicando las mejoras anteriormente mencionadas, para así mejorar la calidad de la aplicación como la experiencia que el usuario pueda tener, para obtener un mejor cuidado de la mascota.

Bibliografía y webgrafía:

En el siguiente apartado se va a mostrar los recursos que se han utilizado para realizar este proyecto:

DRAW.IO – Página web para el diseño de diagramas

<https://app.diagrams.net/>

GanttProject – Aplicación de gestión de proyectos

<https://www.ganttproject.biz/>

ICONSHOCK – Página web de creación de logotipos

<https://www.iconshock.com/?prd=affcomts13820>

Android Studio – Documentación y tutoriales

<https://developer.android.com/studio>

ROOM – Sección de Room en la documentación de Android Studio

<https://developer.android.com/training/data-storage/room?hl=es-419>

MEDIUM – “Android Architecture Components Room - Relationships”

<https://medium.com/android-news/android-architecture-components-room-relationships-bf473510c14a>

YOUTUBE – “Curso ANDROID desde cero” – Definiendo el Adapter #18

<https://www.youtube.com/watch?v=rq92Socn9sE>

NOTIFICACIÓN – Sección developers de Android -> “Cómo crear una notificación”

<https://developer.android.com/training/notify-user/build-notification?hl=es-419>

YOUTUBE – “NOTIFICACIONES PUSH programadas con ALARMMANAGER y BROADCASTRECEIVER”

<https://www.youtube.com/watch?v=TEoe4JTQOEA>

GITHUB – Plataforma para la gestión de repositorios de proyectos

<https://github.com/>

STACKOVERFLOW – Web de dudas y soluciones de programadores

<https://stackoverflow.com/>

Anexos:

- APK de la aplicación:

Junto a lo que es la documentación, se pueden encontrar otros archivos como la apk de la aplicación para probarla en cualquier dispositivo.

- Archivo zip:

Se ha proporcionado el archivo zip que contiene todo el código que se ha realizado para que funcione la aplicación.

- Instalador Android Studio:

Si decide probar la aplicación desde un emulador este proyecto también trae un instalador del editor que se ha usado para realizar la aplicación, puede crear las máquinas virtuales para probar esta aplicación.

- Instrucciones de instalación:

Se explica cómo pasar la que contiene este proyecto a su móvil, y así instalar la aplicación en su dispositivo.

- Memoria en PDF:

Junto a la aplicación, se adjunta la memoria que contiene toda la explicación detallada de la realización del proyecto.

- Repositorio GitHub:

Por último, se ha proporcionado el acceso al repositorio de la aplicación para observar la evolución de la aplicación que ha tenido. También es otra opción para ver el código fuente de la aplicación.

<https://github.com/KevinB00/FidoFriend>