



WILDLIFE ZOO

Nerea Mañez Moya
2º DAM

Indice

JUSTIFICACIÓN DE SELECCIÓN DE PROYECTO.....3

TECNOLOGÍAS UTILIZADAS.....3

DISEÑO.....3

DIFICULTADES.....8

RESULTADOS OBTENIDOS.....9

PROPUESTA DE MEJORA.....9

CONCLUSIÓN.....9

JUSTIFICACIÓN DE SELECCIÓN DE PROYECTO

He seleccionado este proyecto ya que me parecía interesante, existen diversos zoológicos en España y uno de los más conocidos está aquí en Valencia (Bioparc) por lo que me ha parecido un reto crear una aplicación que sería útil para este al menos como estructura básica de la misma.

El tiempo destinado a la búsqueda de información para la población de la base de datos ha sido ciertamente entretenida y me ha ayudado a aprender al mismo tiempo algunos datos respecto a ciertos animales.

TECNOLOGÍAS UTILIZADAS

Utilización de Android Studio como IDE (es el oficial para desarrollo de apps en Android) ya que dicha aplicación se ha desarrollado para dispositivos móviles. Tiene una interfaz relativamente sencilla, se ha utilizado con anterioridad y tiene compatibilidad con GitHub necesaria para la compartición de este proyecto.

Java como lenguaje de programación ya que ha sido el utilizado durante la mayor parte del curso y teniendo en cuenta que existe mayor cantidad de documentación por llevar más años en uso (Kotlin ha presentado problemas en la conexión con la base de datos por lo que después de varias pruebas se decidió pasar a Java debiendo dejar la opción de uso de Kotlin Multiplatform a un lado con este cambio)

Realm (pertenece a MongoDB) para la creación de la base de datos.

Esto genera una base de datos en local, tiene una versión destinada al uso en aplicaciones móviles, se trata de una base de datos orientada a objetos lo que facilita su uso en este caso y al tratarse de un SDK de OpenSource existe bastante documentación al respecto ya que mucha gente parece haberlo usado para diversos proyectos tanto para móvil como para otras plataformas.

DISEÑO

Solo el usuario administrador puede añadir, modificar y eliminar datos de la base de datos por lo que la visualización de las pantallas será diferente dependiendo del usuario con el que accedamos. Todos los usuarios puede visualizar la información pero cada uno verá sus tareas definidas mientras que el administrador podrá verlas todas.

En la base de datos se han creado los diferentes objetos definidos básicos en clase y se les ha añadido a algunos mas que se han considerado necesarios a demás de un objeto nuevo llamado 'Habitat' que define donde se encuentra el animal, este contiene : un id como clave primaria, un tipo (definidos en un Enum) y una descripción.

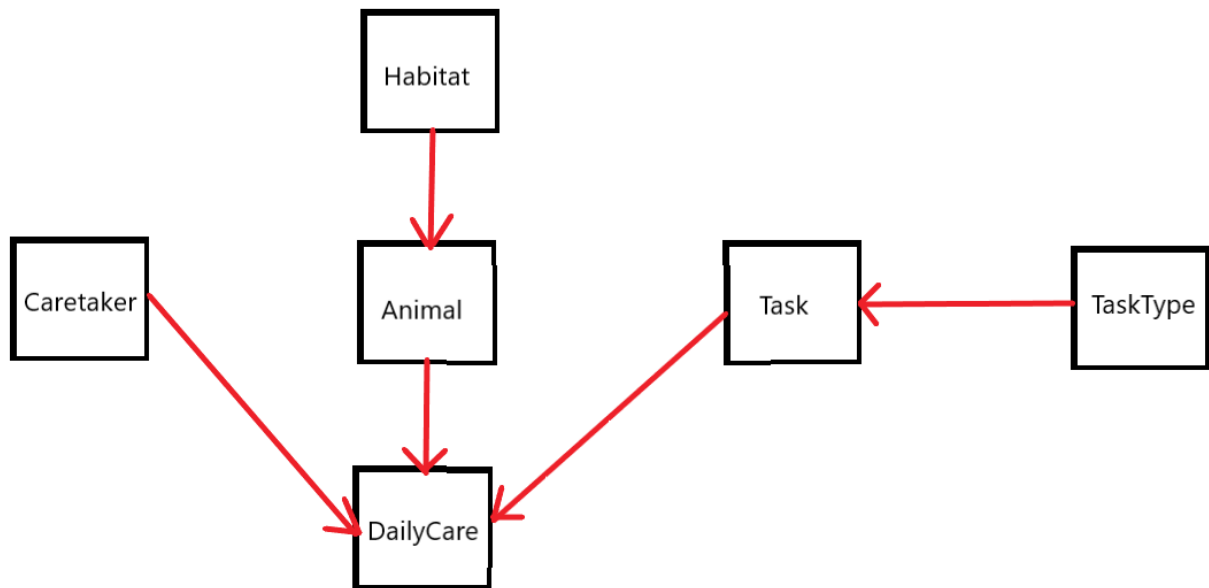


Diagrama final de la base de datos

La aplicación comienza en una pantalla de login pudiendo acceder a una de registro desde esta, ambas con una interfaz muy similar y al completar el login se accede a otra activity con únicamente tres botones que nos llevan a las listas correspondientes.

Wildlife Zoo

Sign In

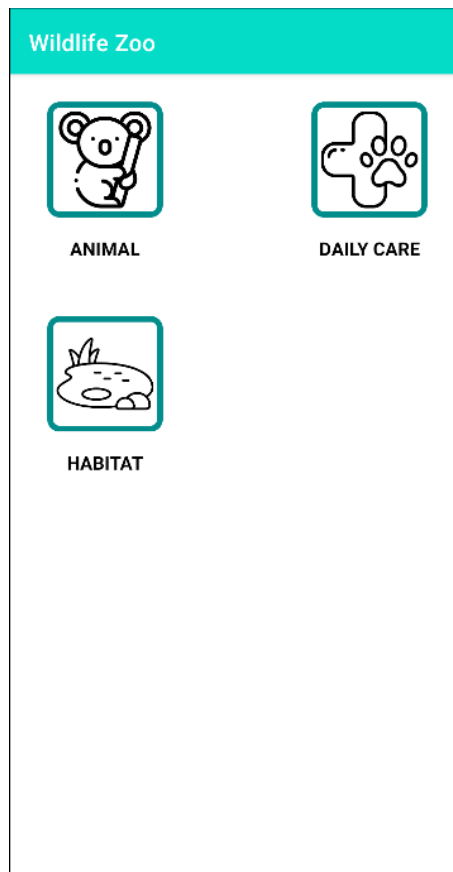
User name

Password

SIGN IN

CREATE AN ACCOUNT

Pantalla de login



Patalla home

Se han usado RecyclerView para mostrar listas de objetos y trabajar sobre los mismos. Para esto ha sido necesario crear un xml para definir la estructura del elemento donde se muestran los datos del objeto y clases extra que se componen de los objetos necesarios para cada lista, finalmente se han usado 4 RecyclerView ya que se ha implementado una más al acceder a la vista de detalle de los hábitats para mostrar los animales que viven en este.

0	savannah
-	
1	jungle
-	
2	forest
	birdcage

Previsualización de datos de habitat, ejemplo de elemento en RecyclerView

HABITAT 5

Habitat type

fresh water

Description

aquarium for amphibians

ANIMALS

16	Perjax	amphibial	Tritón
17	Lyka	amphibial	Tritón
18	Zarpi	amphibial	Ajolote

UPDATE

DELETE

Activity vista de detalle del háitat con id 5

Estas vistas de detalle cuentan con un sistema de doble validación tanto para la eliminación del elemento como para su modificación, tomando como ejemplo los háitats ya que són los que considero más completos comprobamos que no existan animales en el háitat a eliminar antes de dar opción a confirmarlo, también los animales de cada háitat cuentan con acceso a su vista de detalle. Hay que tener en cuenta que un animal nunca podrá er creado si definir su háitat.

Continuando con el ejemplo, han sido utilizados Spinners para la selección de cierta información como es en este caso el tipo de háitat, se han definido en un Enum y se han introducido en dicho Spinner consiguiendo de esta forma impedir que se puedan crear cualquier tipo de haitats, esto también se podría hacer creando un objeto en Realm HabitatType.

La edición de los animales del háitat no se permite en esta vista, no se puede eliminar ni añadir ningún animal al háitat.

HABITAT 5

Habitat type

Description

ANIMALS

16

Perjax

amphibial

Tritón

17

Lyka

amphibial

Tritón

18

Zarpi

amphibial

Ajolote

mountain

fresh water

forest

desert

sea

savannah

jungle

river

✓

✗

Ejemplo spinner en la edición de hábitats

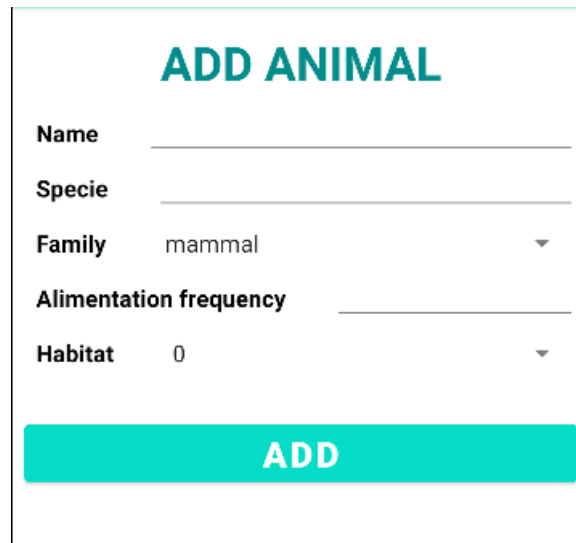
Para finalizar podemos ver como en la vista de lista de animales para cambiar de ejemplo (este tiene más datos que un hábitat) encontramos un botón flotante en la esquina inferior derecha que nos permitirá añadir un nuevo animal a nuestra lista.

0	Nasha	mammal	Jirafa
1	Joy	mammal	Jirafa
2	Duna	mammal	Cebra
3	Also	mammal	Cebra
4	Fifi	mammal	Lémur
5	Lala	mammal	Lémur
6	Bibi	mammal	
7	Louie		



FloatActionButton en animales

Introduciremos todos los datos del animal y luego sobre el botón 'ADD' que taién cuenta con doble validación. Tanto esta activity como las de detalle al eliminar el objeto se cierran de forma automática y nos devuelven a la activity anterior (la lista).



Pantalla añadido de animales

DIFICULTADES

- Al tratar de conectar con la base de datos usando Kotlin se generaba una pila de errores la cual no supe solucionar a pesar de buscar documentación al respecto y probar diversas cosas, por lo que finalmente opté por pasar a Java y este no me dio problemas en ningún momento.
- Existe falta de documentación respecto al uso de claves ajenas en Realm con Java así que opté por solucionarlo creando un campo por cada FK en cada objeto y en el momento de obtener datos usaba estos para realizar un filtro en una búsqueda de realm, esto implica mas líneas de código pero era la forma más rápida de solucionar el problema. Por esto mismo he generado una clase para cada RecyclerView que contenía un objeto de cada uno de los necesarios.
- Los RealmObject parecen no poder serializarse por esto mismo he pasado los ID de estos objetos entre Activities en lugar del objeto completo.
- Para poder editar la información existente en la base de datos debemos añadir `‘.allowWritesOnUiThread(true)’`, de esta forma indicamos que el hilo principal de nuestra aplicación podrá realizar directamente los cambios. No lo veo recomendable si la cantidad de datos es muy grande pero en este caso solo añadimos, modificamos o eliminamos un elemento a la vez lo cual no parará la ejecución del Thread demasiado tiempo.
- A la hora de hacer update o delete sobre un elemento se debe realizar en una transacción, esto inicialmente me ha llevado un tiempo averiguar el correcto funcionamiento ya que toda la información que cambien en el objeto aunque no se ejecute dicho cambio sobre la base de datos debe realizarse en dicha transacción. Es bastante sencillo una vez encuentras la documentación y realizas algunas pruebas fallidas.

RESULTADOS OBTENIDOS

Una vez superadas las dificultades anteriormente nombradas obtenemos como resultado una aplicación bastante funcional y completa a demás de nuevos conocimientos que podrían mejorar la propia aplicación dedicando más tiempo.

En todo caso el control de las tareas de alimentación sería conveniente realizarlo teniendo en cuenta la cantidad de veces que debe ser realizada y poder marcar cada vez que se ha realizado haciendo que el contador disminuya y se complete dicha tarea al llegar el contador a cero.

PROPUESTA DE MEJORA

- Para los hábitats podría mostrarse un mapa interactivo del zoológico de manera que puedas ver la posición de cada uno si fuera necesario. He realizado una búsqueda rápida de información al respecto pero no he encontrado la forma de implementar esto con Java.
- También podría ser una buena implementación la generación de tareas automáticamente teniendo en cuenta la fecha de realización de la última tarea de ese tipo al animal correspondiente y la frecuencia de realización, de esta forma se evitaría la necesidad del administrador a acceder y crear cada una de las tareas. Podría ser simplemente un click que generara las tareas ahorrando mucho tiempo a la persona encargada de esto.

CONCLUSIÓN

A pesar de lo que siento ha sido una falta de tiempo para la realización del proyecto teniendo en cuenta la amplia cantidad de funcionalidades que se podrían implementar en un caso real se trata de una aplicación bastante completa y lo más fiel a la realidad posible.

El uso de Realm es sencillo en comparación con el de otras bases de datos y ofrece gran cantidad de posibilidades para según que tipo de aplicaciones. Dicho esto, falta información respecto a la solución de errores con el uso de Realm con Kotlin pero parece que puede llegar a tener más utilidades que con Java en el futuro.