

## Introducción

Se te proporciona una app funcional con la que hemos trabajado en clase en otras unidades. Presenta un listado de orcos contruidos aleatoriamente y, sobre ese listado puedes realizar ciertas acciones:

- Añadir un orco a la lista (vía opción de menú)
- Borrar un orco de la lista (mediante un 'swipe')
- Quitar energía a un orco (pulsando sobre ese orco)
- Generar una lista nueva (vía opción de menú)

Toda la información que se genera en esta app es volátil, es decir, no se mantiene si abandono la app y vuelvo a a entrar.

Lo que queremos hacer es dotar a los datos de esta app de persistencia, es decir, que pueda salir y volver a entrar trabajando con los datos que tenía en la ejecución anterior.

En la unidad anterior vimos cómo hacer esto usando un almacenamiento local (room → sqllite database). En esta unidad vamos a implementar esa persistencia utilizando un almacenamiento remoto, en un servidor externo, utilizando una REST API, uno de los métodos más ampliamente utilizados en todo tipo de aplicaciones en la actualidad (y no sólo de móviles).

La REST API será creada por nosotros en *mockapi*, y la explotaremos utilizando la librería *Retrofit*.


Para ello:

## Especificaciones

1) Crea en mockapi un recurso que sea un reflejo lo más fiel posible de nuestra clase Orco. Para ello, debido a las peculiaridades de mockapi, tendrás que modificar ligeramente la clase Orco. Puedes crear el recurso en un proyecto nuevo o en uno que ya tengas creado, pero recuerda que mockapi (en su versión gratuita) sólo te permite 1 proyecto y 2 recursos.

2) Puedes prescindir de OrcosProvider, pues ya no la necesitaremos.

3) Modifica la aplicación para que cada vez que el usuario realice una acción (tocar, swipe o una opción de menú) primero se ejecuten los cambios pertinentes en el servidor y después, sólo si todo ha ido bien, se reflejen en la app. Si la acción en el servidor falló, se debe informar de ello al usuario.

Hay un pequeño cambio en la filosofía de la app: el botón  ya no quiere decir que se regenere la lista (no tiene sentido, estamos interactuando con un servidor en lugar de inventarnos los datos) sino que quiere decir “refresca”, es decir, “dame la información actualizada del servidor”. ¿Por qué? Porque varios usuarios podrían estar trabajando con el mismo servidor y podrías no tener la información actualizada.

Por tanto, las acciones deben quedar así:

- Al arrancar la app, debe mostrar todos los orcos que haya en el servidor. Si se produce un error en la recuperación de los datos, debe mostrar el mensaje “Error en la carga de datos”.
- Al pulsar sobre un orco debe modificarse ese orco en el servidor y, si todo ha ido bien, mostrar el mismo cambio en la lista. Si se produce un error al modificar el orco en el servidor, debe mostrar el mensaje “Error al modificar”.
- Al hacer ‘swipe’ sobre un orco debe eliminarse ese orco en el servidor y, si todo ha ido bien, mostrar el mismo cambio en la lista. Si se produce un error al eliminar el orco en el servidor, debe mostrar el mensaje “Error al eliminar”.
- Al pulsar sobre la opción de menú “nuevo” debe insertarse un orco en el servidor y, si todo ha ido bien, mostrar el mismo cambio en la lista. Si se produce un error al insertar el orco en el servidor, debe mostrar el mensaje “Error al insertar”.

- Al pulsar sobre la opción de menú “refrescar” debe hacer exactamente lo mismo que al arrancar la app: mostrar todos los orcos que haya en el servidor. Si se produce un error en la recuperación de los datos, debe mostrar el mensaje “Error en la carga de datos”.

Los mensajes se mostrarán por medio de tostadas.

Recuerda que la tostada requiere de un contexto (en este caso del fragmento en el que se muestra el listado), pero en el viewmodel no debes acceder a elementos de la IU porque el viewmodel sobrevive al ciclo de vida de los elementos del IU. Por ello, te sugiero que para mostrar las tostadas sigas una técnica similar a la que se ha implementado para detectar los cambios que conllevan la actualización del listado.

Ahora bien, ¿cómo puedo simular los errores?

- Para la carga inicial, el “refresco” o la inserción, puedes desconectar los datos y el wifi del dispositivo y obtendrías el error
- Para la modificación y el borrado, con la aplicación en marcha, accede a mockapi y borra los datos de uno de los orcos (equivaldría a que otro usuario ha eliminado ese orco después de que lo recuperases del servidor) → tienes en el listado del dispositivo un orco que no está en el servidor. Si intentas modificarlo o borrarlo debe salirte el error.

#### Entrega

Deberás entregar el proyecto adaptado a las especificaciones proporcionadas arriba y también la “base url” de tu rest api y el nombre del “recurso”. Es decir, la entrega incluye tanto archivo como texto.

#### Puntuación

- Creación de la RESTAPI y adaptación de la clase Orco → **1p**
- Carga inicial y refrescar, incluyendo los cambios en el viewmodel y los métodos del interfaz de retrofit → **2p**
- Modificación, incluyendo los cambios en el viewmodel y los métodos del interfaz de retrofit → **2p**
- Inserción, incluyendo los cambios en el viewmodel y los métodos del interfaz de retrofit → **2p**
- Borrado, incluyendo los cambios en el viewmodel y los métodos del interfaz de retrofit → **2p**
- Gestión de los errores y mensajes → **1p**