Prueba técnica Android A3

1. Información general

Se plantean cuatro ejercicios para los cuales se proporcionan determinadas clases de apoyo incluidas en las carpetas Ejercicio 1, Ejercicio 2, Ejercicio 3 y Ejercicio 4.

La entrega los ejercicios se deberá realizar en un repositorio personal, utilizando el Readme del mismo para elaborar una breve memoria que de respuesta a algunas de las preguntas planteadas y que se podrá usar para justificar las decisiones tomadas en los distintos ejercicios.

2. Ejercicio 1

Dado el código proporcionado en la carpeta Ejercicio 1, responde a los siguientes puntos:

2.1. DESCRIPCIÓN Y RESPUÉSTAS

Escribe una breve memoria explicando el propósito del código que realiza la función *combineOutputs*. En ella, contesta también a las siguientes cuestiones:

- ¿Por qué en el código se usa la operación *.copy*?
- ¿Por qué en el código se usan las operaciones *.apply*?
- ¿Qué ocurriría si se cambiasen por un *.also*?
- ¿Por qué es necesario utilizar la operación *toMutableList*?

2.2. TESTING

Completa los test unitarios que consideres necesarios para verificar el código contenido en la función *combineOutputs*.

2.3. OPTIMIZACIÓN

Realiza un desarrollo alternativo en la función *combineOutputsSimplified* que simplifique y mejore la función *combineOutputs*.

2.4. OPTIMIZACIÓN

Completa la función *combineWith* para que realice lo mismo que la function combineOutputs pero siendo más cercana a nuestro lenguaje natural.

3. Ejercicio 2

Se proporcionan los modelos *Movie* y *Platform* en la carpeta Ejercicio 2 (que representan respectivamente una película y un objeto que identifica las plataformas de streaming donde se puede encontrar una película) y una lista de ambos donde éstos se relacionan por el campo movieId.

3.1. DESARROLLO

Completa la función *getMoviesInPlatforms* para que ésta construya un nuevo modelo llamado *MovieInPlatform* que contenga el agregado de la información que contienen *Movie* y *Platform*. La función *getMoviesInPlatforms* debe omitir aquellas películas que no sean encontradas en ninguna plataforma y proporcionar como salida una lista de objetos *MovieInPlatform*.

3.2. TESTING

Completa todos los test que consideres necesarios para verificar el funcionamiento del código desarrollado.

4. Ejercicio 3

Se proporciona el modelo *UserLoginData* y las clases *UserProcessor* y *ServiceApi* en la carpeta Ejercicio3. Estas clases hacen lo siguiente:

- *UserProcessor*: Realiza un procesamiento **pesado** para obtener un UID para un usuario basado en su información, construyendo al terminar una instancia de *UserLoginData*.
- *ServiceApi*: Simula una llamada a una API externa que procesa el objeto *UserLoginData* y devuelve un valor booleano si ha enviado correctamente la información

Completa el código de la clase *ServiceViewModel* para que, utilizando **corrutinas**, se pueda llamar a la función *processAndLogin* para que coordine el siguiente proceso:

- Obtener a través del *UserProcessor* la información del usuario en el objeto *UserLoginData*.
- Llamar al servicio a través de *ServiceApi* y obtener la respuesta.
- Escribir en el LiveData un mensaje que identifique si el proceso se ha completado para que en la UI el usuario pueda ver ese mensaje

5. Ejercicio 4

En la carpeta Ejercicio 4 se proporciona un test de ejemplo a completar.

5.1. DESARROLLO

Escribe una función en Kotlin que acepte dos valores opcionales de un mismo tipo genérico y una lambda como argumento. La función debe verificar si ambos valores son diferentes de nulo y, en ese caso, ejecutar la lambda devolviendo ambos valores como argumentos de la misma despues de haber verificado su no nulidad.

5.1. TESTING

Completa todos los test que consideres necesarios para verificar el funcionamiento del código desarrollado.