Cliente Api Evernote

ARQUITECTURA

1. Conexión con el Api de Evernote (*paquete .evernoteapi*)

He usado el sdk de Evernote para realizar las llamadas al api de Evernote. He “wrapeado” tanto el cliente como los callbacks de las llamadas, este wrapper debería de heredar de una interfaz que exponga los métodos del api, de modo que si por ejemplo, por alguna razón, dejásemos de usar el SDK de Evernote y realizásemos las llamadas al Api Rest usando Retrofit, esta también heredase del mismo interfaz. Por tanto la parte relativa al negocio propio de la aplicación solo conociese el interfaz y no si la implementación se ha realizado de una manera o de otra, sería totalmente transparente e independiente de cambios de implementación.

Pasa exactamente lo mismo con los callbacks, se usan callbacks propios para que la parte del negocio no dependa de la implementación de la conexión con el Api.

1. Pantalla de Login (*paquete .login*)

Al haber optado por usar el SDK que proporciona Evernote, este proporciona un método de login que lo que hace es lanzar una Activity propia con un webview en el que el usuario se logea, y proporciona los permisos necesarios para realizar el resto de llamadas.

En caso de que esta solución no fuera válida por razones estéticas u otras razones mas técnicas, se realizaría una implementación usando retrofit e implementando la autenticación por medio de oAuth2.

1. Listado de Notas (*paquete .notes*)

Para el listado de notas he decidido usar recyclerView en lugar del Listview al uso, esto hace el listado mucho mas fluido y usa parte del Android binding, aunque no se usa en este caso el binding del xml ya que en los recyclerview tiene un comportamiento extraño aun, normalmente cuando en los cardview se usan imágenes y algún tipo de librería como Picasso o Volley para cargarlas. Aun no siendo el caso he decidido no usarlo. Si lo usáramos estaríamos usando una arquitectura MVVM para esta parte de la aplicación mientras que en este caso podría ser un hibrido entre MVC y MVVM.

Otra opción hubiera sido usar MVP con Dagger2 e inyección de dependencias, pero personalmente me parecía matar moscas a cañonazos.

He añadido gestión de paginación en el listado y swipeToRefresh para refrescar el listado, así como una sencilla gestión de los errores para poder reintentar en caso de falta conexión o fallo casual en alguna llamada.

TEST UNITARIOS

Aunque no hay excesivo negocio en la aplicación que testear de manera unitaria. Se ha testeado el adapter como ejemplo de cómo se debería hacer con la aplicación. Igualmente podría testearse el UI mediante Espresso.

DEPENDENCIAS

Además de las librerías propias de soporte que ofrece Android, he usado las siguientes dependencias:

* **Butterknife ( com.jakewharton:butterknife:7.0.1 )**

Se usa para la inyección de vistas y eventos sobre ellas en la aplicación.

* **SDK Evernote (com.evernote:android-sdk:2.0.0-RC3 )**

Facilita la conexión con el Api de Evernote

COMENTARIOS

No he realizado el punto 6, en el cual se debía recoger escritura a mano e inspeccionarlo mediante un OCR dado que no me ha dado tiempo, y he considerado que era el punto que mas podía considerar como “extra”, dado que investigando no he visto una manera limpia de realizarlo en tan poco tiempo, hacerlo bien significaría implementarlo yo mismo y llevaría más tiempo del que se da para terminar la prueba.