UNIVERSIDAD LAS PALMAS DE GRAN CANARIA

PROGRAMACIÓN DE APLICACIONES MÓVILES NATIVAS

INFORME SEMANA 8:

RECOMENDACIONES DE ARQUITECTURA PARA APLICACIONES ANDROID

PRESENTADO POR: IVONNE YULIETH OJEDA CUERVO

AZ930245

LAS PALMAS DE GRAN CANARIA – 2023

RECOMENDACIONES DE ARQUITECTURA PARA APLICACIONES ANDROID

Después de realizar la revisión de las recomendaciones para la arquitectura de Android se escogieron las siguientes 5:

1. Usa una capa de datos claramente definida.

Descripción: "La capa de datos expone los datos de la aplicación al resto de la app y contiene la gran mayoría de su lógica empresarial.

- Debes crear repositorios incluso si contienen una sola fuente de datos.
- En las apps pequeñas, puedes colocar tipos de capas de datos en un módulo o paquete data." [1]

Decisión: Si, será incluida en el proyecto.

Justificación: Se incluirá en el proyecto, ya que se ha adoptando una arquitectura MVVM/Clean architecture. Esta capa de datos actúa como un punto central para exponer datos. Incluso si nuestro proyecto es relativamente pequeño, es fundamental crear repositorios para abstraer la fuente de datos API y BD. Al hacerlo, aseguramos que nuestra aplicación sea altamente modular y fácil de mantener. La capa de datos bien definida proporciona una separación clara de responsabilidades, lo que es esencial para garantizar la escalabilidad y la legibilidad del código en un entorno de desarrollo MVVM/Clean architecture..

2. Usa una capa de la IU claramente definida.

Descripción: "La función de la IU es mostrar los datos de la aplicación en la pantalla y servir como punto principal de interacción con el usuario. Cuando los datos cambian, ya sea debido a la interacción del usuario (como cuando presiona un botón) o una entrada externa (como una respuesta de red), la IU debe actualizarse para reflejar los cambios. En efecto, la IU es una representación visual del estado de la aplicación tal como se recuperó de la capa de datos". [2]

Decisión: Si, será incluida en el proyecto.

Justificación: Al tener una capa de IU bien definida, estamos siguiendo los principios de las arquitecturas MVVM (Model-View-ViewModel) y Clean, ya que estas enfatizan la separación de preocupaciones y la modularización del código. Esta separación de responsabilidades y claridad en la capa de IU es fundamental para garantizar que el código sea mantenible, escalable y fácil de comprender.

3. La capa de datos debería exponer los datos de la aplicación mediante un repositorio.

Descripción: "Los componentes de la capa de la IU, como los elementos componibles, las actividades o los ViewModels, no deben interactuar de forma directa con una fuente de datos" [1]. " La capa de datos está

formada por repositorios que pueden contener de cero a muchas fuentes de datos. Debes crear una clase de repositorio para cada tipo de datos diferente que administres en tu app. Por ejemplo, puedes crear una clase MoviesRepository para datos relacionados con películas o una clase PaymentsRepository para datos relacionados con pagos" [3].

Decisión: Si, será incluida en el proyecto.

Justificación: Al utilizar un repositorio, aseguramos que los componentes de la IU no dependan directamente de una fuente de datos específica. Esto hace que sea más fácil cambiar o agregar fuentes de datos en el futuro sin afectar a la lógica de la IU. Además, facilita las pruebas unitarias al permitir la creación de implementaciones de repositorio ficticias o simuladas para pruebas.

4. Usa corrutinas y flujos.

Descripción: "Usa corrutinas y flujos para establecer la comunicación entre capas. Usa corrutinas y flujos. ViewModel interactúa con los datos o las capas de dominio mediante lo siguiente:

- Flujos de Kotlin para recibir datos de aplicaciones.
- Funciones suspend para realizar acciones con viewModelScope." [1]

Decisión: Si, será incluida en el proyecto.

Justificación: Se utilizarán corutinas y flujos en nuestra aplicación, ya que esto hará que la app sea más eficiente. Esto se debe a que las corutinas y flujos permiten realizar diferentes tareas al mismo tiempo sin ralentizar la aplicación, lo que es una gran ventaja. Además, ayudan a mostrar información en tiempo real en la pantalla, lo que hace que la app sea más atractiva para los usuarios. También facilitan la comprensión del código. Por último, estas herramientas garantizan que la aplicación sea segura y no tenga problemas con la memoria, lo que es esencial para que funcione correctamente.

5. Usa una capa de dominio.

Descripción: La capa de dominio es responsable de encapsular la lógica empresarial compleja o la lógica empresarial simple que varios ViewModels reutilizan. Esta capa es opcional porque no todas las apps tendrán estos requisitos. Solo debes usarla cuando sea necesario; por ejemplo, para administrar la complejidad o favorecer la reutilización.[4]

Decisión: Si, será incluida en el proyecto.

Justificación: Aunque el proyecto puede ser de menor escala, la capa de dominio aporta claridad y organización a la lógica empresarial. Adicionalmente, al separar la lógica empresarial en una capa de dominio, seguimos las mejores prácticas de arquitectura, lo que facilita futuras expansiones y modificaciones del proyecto, garantizando una base sólida incluso en aplicaciones de menor tamaño.

BIBLIOGRAFÍA

[1]“Recommendations for Android architecture,” Android Developers. https://developer.android.com/topic/architecture/recommendations?hl=es-419%E2%80%8B (accessed Oct. 31, 2023).

[2]“Capa de la IU | Desarrolladores de Android,” Android Developers. https://developer.android.com/jetpack/guide/ui-layer?hl=es (accessed Oct. 31, 2023).

[3]“Capa de datos | Desarrolladores de Android,” Android Developers. https://developer.android.com/jetpack/guide/data-layer?hl=es (accessed Oct. 31, 2023).

[4]“Capa de dominio | Desarrolladores de Android,” Android Developers. https://developer.android.com/jetpack/guide/domain-layer?hl=es (accessed Oct. 31, 2023).