ULPGC Universidad de Las Palmas de Gran Canaria | Escuela de Ingeniería Informática

Recomendaciones de arquitectura para aplicaciones Android

Programación de Aplicaciones Móviles Nativas

Autores: Fernando Sanfiel Reyes

Daniel Betancor Zamora

Pablo Santana Susilla

Fecha: 02/11/2023

Índice

1.- Introducción

La arquitectura de una aplicación móvil es el esqueleto fundamental sobre el que se construye toda la funcionalidad y experiencia del usuario. Por eso mismo, vamos a investigar en la página oficial para desarrolladores de Android las recomendaciones de arquitectura propuestas. Posteriormente, elegiremos 5 de las recomendaciones que se consideran relevantes para la elección de una arquitectura.

2.- Elección de las 5 recomendaciones más relevantes

Arquitectura en capas

Descripción:

La arquitectura en capas es un enfoque que implica la división de una aplicación en capas o niveles separados, cada uno con su responsabilidad específica. Las capas típicas incluyen la capa de presentación (interfaz de usuario), la capa de lógica de aplicación y la capa de datos. Esta división ayuda a organizar el código de manera modular y separar las preocupaciones, lo que facilita la comprensión, el mantenimiento y la escalabilidad del proyecto.

Básicamente, el principio fundamental de la arquitectura en capas es la separación de problemas. Esto significa que debemos dividir nuestra aplicación en diferentes capas, cada una de las cuales se encarga de una tarea específica.

Decisión:

Está planteado implementar esta recomendación en el proyecto de la asignatura, ya que uno de los principios de la arquitectura escogida (*Clean Arquitecture*) es la separación en capas de los diferentes módulos de la aplicación.

Justificación:

La arquitectura en capas es una forma efectiva de organizar las aplicaciones nativas de Android. Al seguir los principios de la arquitectura en capas, podemos crear aplicaciones que sean robustas, mantenibles y fáciles de extender.

Además, Las capas aisladas permiten probar componentes de manera individual, lo que garantiza una mayor calidad del software.

ViewModel

Descripción:

Los *ViewModels* nos permiten proporcionar diferentes estados de la UI y un acceso modular a la capa de datos, asegurando así la integridad de la capa de datos impidiendo posibles brechas de seguridad.

Decisión:

Teniendo en cuenta la gestión de usuarios que debemos implementar en nuestra aplicación, se ha decidido seguir esta recomendación e implementarla durante el desarrollo de nuestra aplicación. Esto es debido a que nos permitirá garantizar la integridad y confidencialidad de los datos personales de nuestros usuarios.

Justificación:

Esta decisión se ha tomado principalmente porque nos permitirá garantizar la integridad y confidencialidad de los datos personales de nuestros usuarios, así como evitar la inserción de datos erróneos o corruptos a la base de datos por parte de usuarios o posibles atacantes.

Además, los *ViewModels* permiten separar la lógica de presentación de las actividades y fragmentos, lo que hace que el código sea más limpio y mantenible.

Convenciones de nombres

Descripción:

A la hora de desarrollar el código, el equipo de desarrollo debe llegar a un acuerdo de convención de nombres para la definición de clases, métodos etc.

Decisión:

Está recomendación nos ha parecido imprescindible teniendo en cuenta que la aplicación será desarrollada por tres programadores que desarrollaran las diferentes capas de la aplicación.

Justificación:

Debido a que todos los programadores podrán implementar cualquier capa de la aplicación, hacer uso de una convención de nombres facilitará el trabajo para cualquiera que deba continuar el trabajo de otro compañero.

Control de dependencias

Descripción:

El control de dependencias, y en especial la inyección de dependencias, es una técnica ampliamente utilizada en la programación. Esto puede proporcionar varias ventajas como la reutilización de código, facilidad de refactorización y facilidad de prueba.

Decisión:

En el equipo de desarrollo se ha decidido implementar esta recomendación en cuanto a la arquitectura, ya que es algo que la *Clean Arquitecture* favorece en gran medida debido a la naturaleza de dicha arquitectura.

Justificación:

Esta decisión ha sido tomada debido a las amplias ventajas que nos aporta la inyección de dependencias en lo que a escalabilidad y mantenimiento se refiere. Además de esto, también favorece mucho a la ampliación en un futuro de la app si se realiza una correcta inyección de dependencias.

Ciclo de vida

Descripción:

Esta recomendación hace referencia al ciclo de vida tanto de la app como de sus actividades y fragmentos, poder hacer control de este, y realizar acciones en función del ciclo de vida de algunas actividades o fragmentos de la aplicación.

En Android, es esencial gestionar adecuadamente el ciclo de vida de las actividades y fragmentos. Cada componente de la interfaz de usuario (UI) pasa por diferentes estados, como "creado", "iniciado", "reanudado" y "detenido", durante su ciclo de vida. Es importante asegurarse de que los recursos se liberen y las acciones necesarias se realicen en cada uno de estos estados para garantizar un comportamiento consistente y una buena experiencia de usuario.

Decisión:

El equipo ha decidido tener en cuenta el ciclo de vida de las actividades realizadas, ya que puede haber usuarios conectados simultáneamente realizando diferentes actividades con sus respectivos ciclos de vida.

Justificación:

Se ha decidido tomar en cuenta el ciclo de vida, ya que nuestra aplicación hará uso de una API de terceros para la consulta de los datos de los diferentes productos alimenticios. Será crucial controlar el ciclo de vida tanto de esta actividad como de las diferentes operaciones de los usuarios con nuestra base de datos.

Por otra parte, la gestión adecuada del ciclo de vida ayuda a evitar la pérdida de datos y permite que la aplicación retome su estado anterior después de eventos como cambios de orientación de pantalla.

Además, permite liberar recursos como conexiones de red o bases de datos, cuando ya no son necesarios, lo que mejora el rendimiento y la eficiencia de la aplicación.

3.- Repositorio de GitHub

https://github.com/PAMN2023/ArchitectureRecommendations.git

4.- Bibliografía

Android. (30 de 10 de 2023). *Desarrolladores de Android*. Obtenido de Desarrolladores de Android: https://developer.android.com/topic/architecture/recommendations?hl=es-419