UNIVERSIDAD LAS PALMAS DE GRAN CANARIA

PROGRAMACIÓN DE APLICACIONES MÓVILES NATIVAS

DISEÑO ARQUITECTÓNICO DE UNA APLICACIÓN

PRESENTADO POR: IVONNE YULIETH OJEDA CUERVO

AZ930245

LAS PALMAS DE GRAN CANARIA

OCTUBRE 2023

TABLA DE CONTENIDO

DISEÑO ARQUITECTÓNICO DE UNA APLICACIÓN

El diseño arquitectónico en una aplicación móvil nativa es un componente crítico para garantizar su eficiencia y funcionalidad. Esta fase de desarrollo define cómo los diferentes componentes de la aplicación interactúan entre sí y cómo se gestionan los recursos del dispositivo. En los siguientes apartados se describirá en detalle el diseño arquitectónico de la aplicación móvil nativa que se desarrollará como proyecto de grado de la asignatura.

1. Definición del Dominio

Siguiendo la línea del informe anterior se continuará con la aplicación de escáner de ingredientes presentes en productos cosméticos.

Funcionalidad principal: Permitir a los usuarios escanear códigos de barras o ingredientes de productos cosméticos para acceder a información detallada, que incluye un resumen rápido cuyo contenido indica la presencia o ausencia de parabenos, sulfatos, alcohol, siliconas y alérgenos de la UE, así como la idoneidad para el acné fúngico. También detalla los efectos esperados de los ingredientes destacados, relaciona ingredientes con tipos de piel y proporciona una calificación de seguridad según el Environmental Working Group (EWG) y el Cosmetic Ingredient Review (CIR). La aplicación presenta una lista completa de ingredientes con sus funciones y notas de calificación.

Las funcionalidades secundarias incluyen:

- Reseña de Productos: Usuarios registrados pueden compartir reseñas de productos para ofrecer información sobre el rendimiento y calidad de los productos.
- Buscador de Productos: Permite a los usuarios buscar productos por nombre o palabra clave, mostrando una lista de productos que cumplen con el criterio ingresado, desde los cuales pueden acceder a la información detallada que ofrece la funcionalidad principal.
- Recursos: Proporciona información relacionada con el cuidado facial y la salud de la piel a través de artículos, enlaces a sitios web y videos.

2. Elementos de la Arquitectura

La arquitectura seleccionada es MVVM en combinación con Clean Architecture siguiendo las recomendaciones de Google para el desarrollo de aplicaciones en Android y al hecho de que este ofrece muchas herramientas que ayudan a implementarla fácilmente durante el proceso de desarrollo. Por otra parte, es una combinación de arquitecturas frecuentemente utilizada por la comunidad de desarrollo de aplicaciones nativas en los últimos años por los beneficios que estas ofrecen, los cuales se detallarán en las secciones finales de este informe.

La arquitectura se compone de las siguientes capas:

1. Presentación: Representa la interfaz de usuario de la aplicación. Se encarga de mostrar la información al usuario y recopilar acciones del usuario, como captura de imágenes y la interacción con la cámara del dispositivo, búsquedas de productos y reseña, etc.
2. Dominio: La capa de Dominio gestiona las reglas de negocio y la lógica interna. En ella, se definen las Entidades que representan la estructura de datos central y los Casos de Uso que permiten la interacción con el escaneo de ingredientes y otras funciones.
3. Datos: Gestionad el flujo de datos y recuperar información de diversas fuentes, como bases de datos, servicios web y archivos. Además, se encarga de la persistencia de datos y facilita el acceso a la información.

Capa de Presentación

- Vistas (View): En esta capa, se encuentran las actividades o fragmentos de la aplicación móvil y que deberán corresponder en cierto modo con el diseño inicial de pantallas planteado en el informe anterior. No debe contener ninguna otra lógica aparte de la lógica de vista (mostrar/ocultar elementos, escuchar eventos de componentes, etc.). Observan los datos del Modelo de la vista (ViewModel) y modifican los componentes en consecuencia.

- Modelo de la Vista (ViewModel) : Actúa como mediador entre las vistas y la Capa de Dominio. Contiene los ViewModels y cualquier lógica relacionada con la presentación, tiene propiedades sincronizadas con la vista

Capa de Dominio

- Casos de Uso (Use cases): Son el reflejo de las interacciones del usuario. Cada caso de uso es una acción. Para nuestro caso algunos de ellos serían:

    - Escaneo de Ingredientes:
        - Captura de Imagen: Permite a los usuarios tomar una foto o cargar una imagen de un producto cosmético.
        - Análisis de Ingredientes: Procesa la imagen capturada para identificar y extraer los ingredientes del producto.
        - Resumen de Ingredientes: Genera un resumen de los ingredientes y propiedades del producto.
    - Reseñas de Productos:
        - Crear Reseña: Permite a los usuarios registrados crear y publicar sus reseñas sobre productos.
        - Ver Reseñas: Permite a los usuarios ver las reseñas de otros usuarios sobre un producto en particular.
    - Búsqueda de Productos:
        - Búsqueda por Nombre: Permite a los usuarios buscar productos ingresando el nombre del producto en un campo de búsqueda.
        - Lista de Resultados: Muestra una lista de productos que coinciden con el término de búsqueda.
        - Detalles del Producto: Muestra información detallada sobre un producto seleccionado, incluyendo ingredientes, calificaciones de seguridad y más.

- Entidades (Entities) : Son objetos de dominio que encapsulan la lógica de negocio y representan los conceptos fundamentales del dominio del problema. Ejemplos:
    - Producto Cosmético: Esta entidad podría incluir información como el nombre del producto, la marca, el código de barras y otros detalles relevantes. Es la entidad principal que se escanea y analiza.
    - Ingrediente Cosmético: Representa un ingrediente específico dentro de un producto cosmético. Incluye detalles como el nombre del ingrediente, su función cosmética, propiedades y posibles efectos en la piel.
    - Reseña de Producto: Una entidad que almacena información sobre las reseñas de los usuarios, incluyendo el producto cosmético, el texto de la reseña, la calificación y el autor de la reseña.
    - Usuario: Representa a los usuarios de la aplicación y puede incluir información de perfil, como nombre de usuario, dirección de correo electrónico, etc.

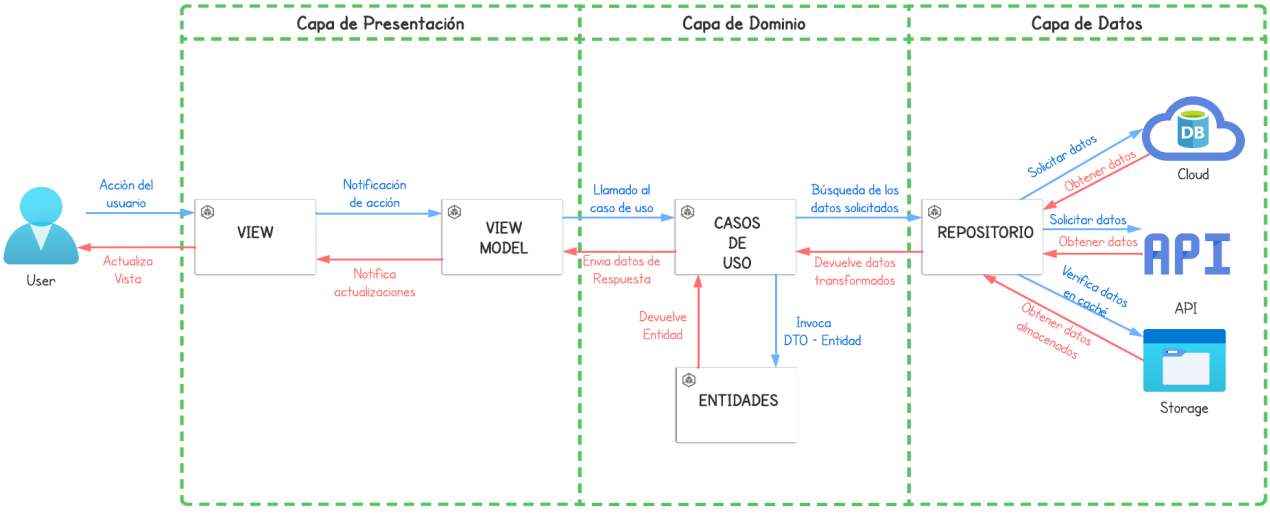
Capa de Datos

- Repositorios: Actúan como puntos de acceso a los datos y se utilizan para interactuar con diversas fuentes de datos, como bases de datos locales, servicios web o almacenamiento en archivos. El mapeo entre entidades y modelos de datos generalmente tiene lugar esta capa.

- Fuentes de datos: Implementan los detalles de nivel bajo de acceso a los datos, como abrir y cerrar conexiones de base de datos, ejecutar consultas y realizar solicitudes HTTP. Las fuentes de datos que usaremos en la aplicación básicamente serán 3:
    - El dispositivo móvil con una copia de los datos de usuario más relevantes.
    - Una BD alojada en la nube que será el centro de datos de la aplicación.
    - BD Externas o API que retornen información sobre ingredientes cosméticos.

3. Diseño de la Arquitectura

A continuación, se presenta el diagrama general de la arquitectura de la aplicación:



Para tener una idea visual de cómo el patrón de diseño MVVM puede aplicarse a Clean Architecture se presenta el siguiente diagrama.

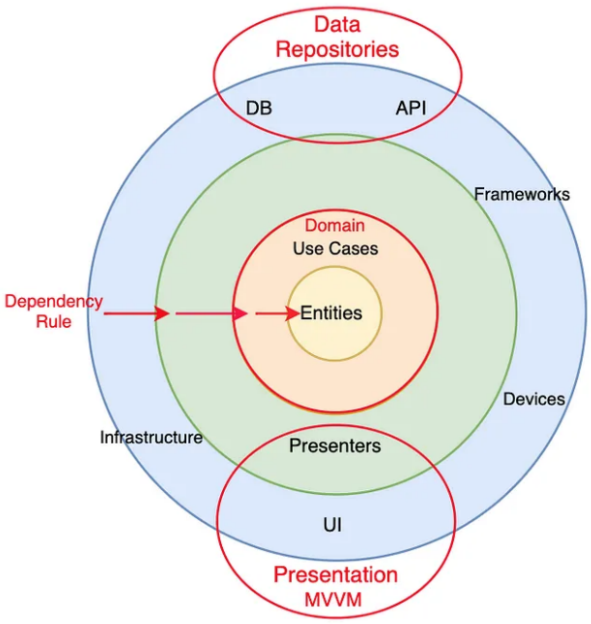


Imagen tomada de "Clean Architecture and MVVM on iOS: Ver

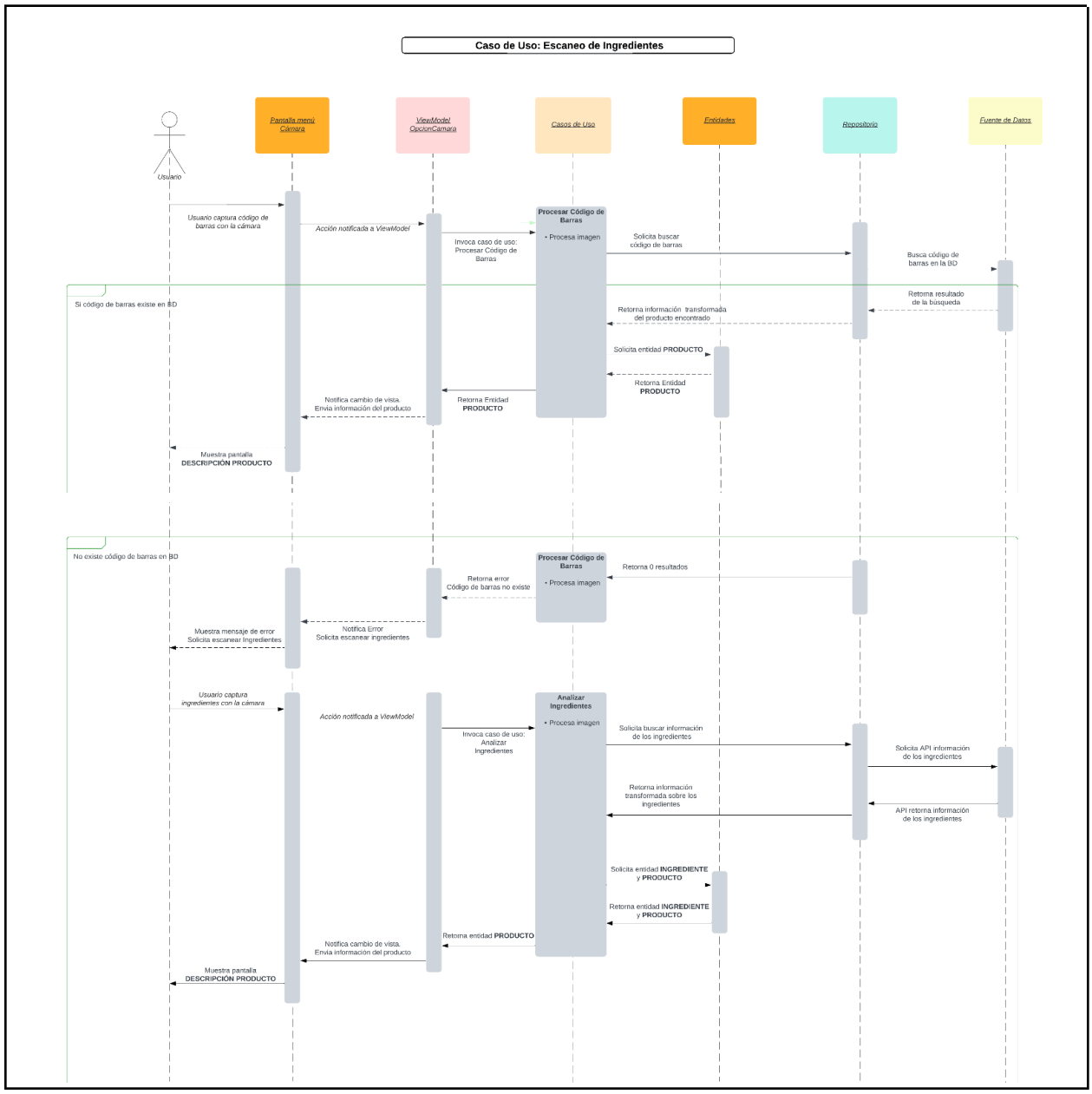
4. Ejemplo Caso de Uso

Caso de Uso: "El usuario puede escanear el código de barras de un producto cosmético y la aplicación retornará la descripción detallada de los ingredientes del producto, en caso de que el producto no exista en la BD de la aplicación se emitirá un error al usuario y se solicitará la captura de los ingredientes del producto para su análisis"

Para implementar este caso de uso se puede seguir el flujo a través de la arquitectura de la siguiente manera:

1. El usuario escanea el código de barras de un producto cosmético en la aplicación móvil.
2. El caso de uso "Procesar Código de Barras" verifica si el código de barras está presente en la base de datos local haciendo una petición al repositorio. Si está presente, se obtiene la información del producto, se transforma de acuerdo a la Entidad Producto (internamente relacionada con la entidad Ingrediente) y se devuelve al ViewModel el cual se encarga de actualizar la vista para que el usuario vea la información retornada.
3. Si el código de barras no se encuentra en la base de datos local, se solicita al usuario tomar una captura de la lista de ingredientes del producto.
4. El caso de uso " Análisis de Ingredientes " procesa la imagen para obtener los ingredientes luego solicita al repositorio consultar los ingredientes, el repositorio se comunica con la API externa para analizar los ingredientes y obtiene la información detallada.
5. La información obtenida se recibe y transforma de acuerdo a las entidades PRODUCTO e INGREDIENTE y se devuelve al ViewModel el cual se encarga de actualizar la vista para que el usuario vea la información retornada.

El siguiente diagrama muestra de manera visual el flujo del proceso.



5. Conclusiones

De acuerdo al análisis realizado se puede concluir que una arquitectura MVVM/Clean architecture proporciona mejores ventajas en el desarrollo de la aplicación por las siguientes razones:

1. Separación de Responsabilidades:

   MVVM/Clean Architecture separa de manera clara las responsabilidades de la lógica de presentación y el negocio, lo que mejora la claridad y mantenibilidad.

   MVI y MVP pueden mezclar estas responsabilidades; en MVI, los intents (intenciones) pueden contener lógica de presentación, y en MVP, el presenter a menudo involucra tareas relacionadas con la vista y la lógica de negocio (esto puede deberse a la falta de una capa de dominio clara en el patrón), lo que puede llevar a la mezcla de responsabilidades.

2. Reutilización de Código:

   MVVM/Clean Architecture permite la reutilización de ViewModels, reduciendo la duplicación de código y mejorando la eficiencia del desarrollo.

   MVI y MVP pueden carecer de esta reutilización; en MVI, la lógica de manejo de intents y reducers a menudo es específica de una pantalla, y en MVP, los presenters pueden ser específicos de una vista, lo que limita la reutilización en otros contextos.

3. Pruebas Unitarias Simplificadas:

   En MVVM/Clean Architecture la separación de capas facilita las pruebas unitarias de la lógica de negocio y de presentación.

   MVI y MVP pueden presentar desafíos en cuanto a las pruebas porque: en MVI, las pruebas de intents y reducers pueden ser complicadas, y en MVP, las pruebas pueden requerir la manipulación de la vista directamente, lo que dificulta la creación de pruebas unitarias efectivas.

6. Bibliografía

[1]Z. wnsxor1993, “MVVM + Clean Architecture 정리,” Velog.io, 2022. https://velog.io/@wnsxor1993/MVVM-Clean-Architecture-%EC%A0%95%EB%A6%AC (Accedido Oct. 14, 2023).

[2]A. Raj, “MVVM Clean Architecture Pattern in Android with Use Cases,” Medium, Jun. 26, 2023. https://medium.com/@ami0275/mvvm-clean-architecture-pattern-in-android-with-use-cases-eff7edc2ef76 (Accedido Oct. 14, 2023).

[3]“Model View View-Model (MVVM).” [Online]. Disponible: https://wit-hdip-computer-science.github.io/semester-2-ent-web-development/topic-16-aurelia-2/talk-1-ui-patterns/ui-patterns.pdf (Accedido: Oct. 14, 2023)

[4]O. Kudinov, “Clean Architecture and MVVM on iOS,” Medium, Mar. 02, 2023. https://tech.olx.com/clean-architecture-and-mvvm-on-ios-c9d167d9f5b3 (Accedido Oct. 14, 2023).

[5]H. Abbas, “Building Android Apps with MVVM and Clean Architecture,” Medium, Jun. 08, 2023. https://medium.com/@hussainabbas365/building-android-apps-with-mvvm-and-clean-architecture-5f123baa24a0 (Accedido Oct. 14, 2023).

[6]“MVVM architecture, ViewModel and LiveData (Part 1),” www.linkedin.com. https://www.linkedin.com/pulse/mvvm-architecture-viewmodel-livedata-part-1-hazem-saleh/ (Accedido Oct. 14, 2023).