Mobile candidate

Marcos Bazzano

Decisiones de arquitectura y Diseño

Para el desarrollo de la aplicación pedida, creí conveniente tomar las diferentes decisiones desde la perspectiva de una aplicación como la de MeLi, una aplicación grande con muchos años de antigüedad y previsible actualización en features a futuro.

Selección del UI Framework

La primera decisión a tomar fue la selección de un framework para la UI entre SwiftUI y UIKit. Dado el tiempo que lleva la aplicación de Mercado Libre en la AppStore y que SwiftUI recién salió en 2019, es sensato asumir que una gran parte de la aplicación fue realizada utilizando UIKit, lo cual favorece enormemente a este para facilitar la integración con las partes ya existentes. A su vez, SwiftUI no permite que la aplicación sea utilizada por algunas versiones más antiguas de iOS ya que no es compatible con versiones de iOS anteriores a la 13, lo cual disminuye el rango de clientes al que está expuesta nuestra aplicación. Estas razones son más que suficientes para tomar la decisión de que **UIKit sea nuestro UI Framework seleccionado.**

Diseño y patrones

Una vez seleccionado el framework, tenemos que decidir cómo diseñaremos nuestra aplicación. Dado que elegimos UIKit como framework, la elección aparentemente obvia es diseñar bajo el patrón Model-View-Controller o MVC, dado que UIKit está basado en el mismo.

Al utilizar este patrón es necesario tener en cuenta que al crecer y complejizarse la aplicación, el controlador puede sobrecargarse ya que es donde se encuentra la mayor parte de las responsabilidades y comunicación entre las partes del sistema. Aunque aún no es necesario dado el tamaño actual de la aplicación, se podría remediar este problema en un hipotético crecimiento utilizando complementariamente el patrón de diseño Coordinator para disminuir la carga del controlador.

Aclaraciones

Sobre el carrusel

Al realizar la pantalla de detalle de un producto seleccionado, realicé un carrusel como el que hay en la aplicación móvil de MeLi. Al querer poblar el carrusel con imágenes, noté que necesitaba un token para acceder al endpoint de get por id, lo cual impedía que accediera a los atributos individuales del producto. Dado que la búsqueda por término sólo traía la imágen miniatura, esto me dejó con dos opciones:

1. Popular el carrusel con la imágen miniatura repetidas veces
2. Popular el carrusel con imágenes *dummy*
3. Que se me proveyera con una token temporal (poco probable)

Luego de plantear este problema en un correo, terminé decidiéndome por la opción 1. dado que de esta forma mostraba que estaba efectivamente utilizando las imágenes de ese producto.

El problema que tiene esta solución es que la imágen es, como mencioné previamente, una miniatura, con tamaño acorde a tal. Esto causa que no sea posible agrandar la imagen al tamaño del carrusel sin que se vea borrosa, independientemente de que contentMode le asigne.

Sobre los atributos de los productos

Dado el posible acceso únicamente al get por término de búsqueda y no de un producto individual, era necesario traer todos los atributos que quería mostrar en la pantalla de detalle de producto en la misma request inicial donde hacía la búsqueda para todos los productos. Esto significa que es mucho más exigente tanto computacionalmente como para la memoria mientras más atributos quiera mostrar.

Elegí mostrar atributos que son comunes a todos los productos y aparentemente obligatorios (digo aparentemente porque fue verificado por revisión manual de las respuestas de la API y un poco de sentido común, dado que no encontré o no hay un endpoint que me de un listado de ellos como los hay para las categorías), dado que para realizar un desglose entero de los atributos del producto como se hace en la app de MeLi, se requeriría mostrar

condicionalmente distintas vistas dependiendo de la categoría del producto, lo cual considero se va del alcance de lo que se espera para esta instancia de evaluación.