

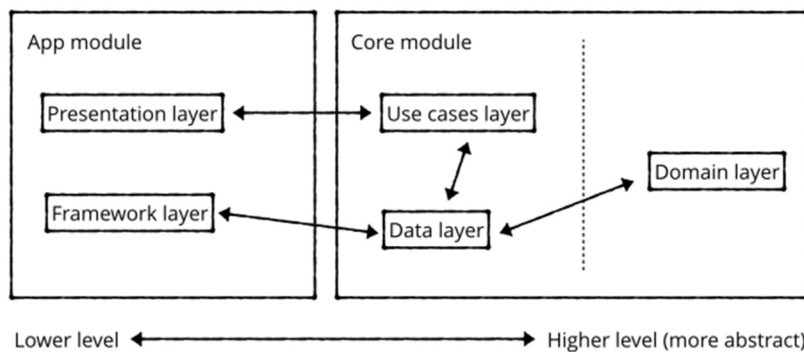
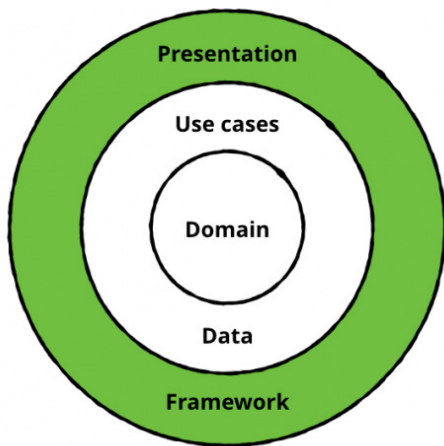
Documentación Challenge

El challenge se realizó de la siguiente manera.

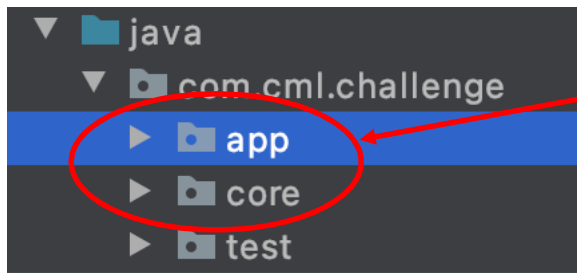
Se tomó como base Clean Architecture y en MVVM para la aplicación de Android ya que suelen ser proyectos grandes.

Las ventajas que nos da son:

- Código más testeable
- Código desacoplado
- Más entendible y mantenible

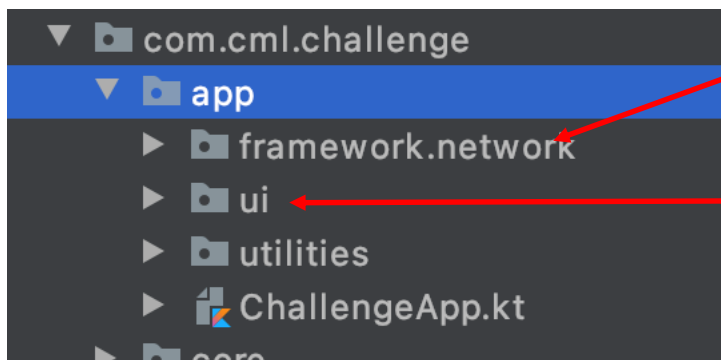


La idea es tener bien separado el core del framework y presentación, por lo que en el proyecto del challenge encontraremos lo siguiente:



En el challenge la separación se realizó por paquetes

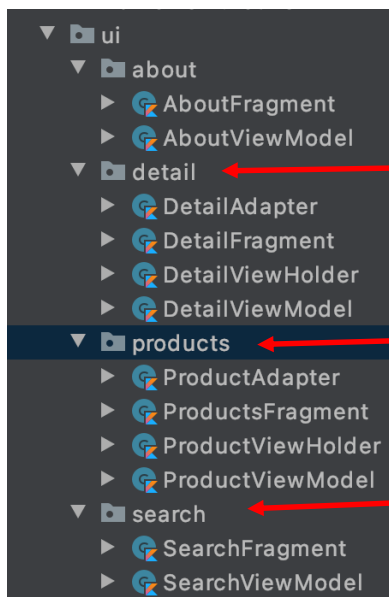
En **app** encontramos todo lo relacionado presentación y framework como sigue:



Framework

Presentación

En **ui** encontramos lo relacionado a la vista, para ellos se utiliza la arquitectura MVVM.

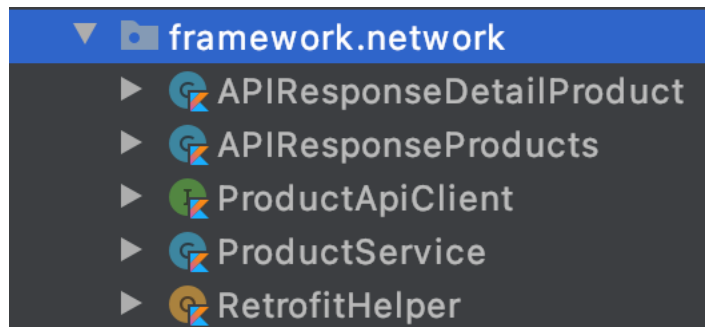


Se encarga de mostrar el detalle del producto

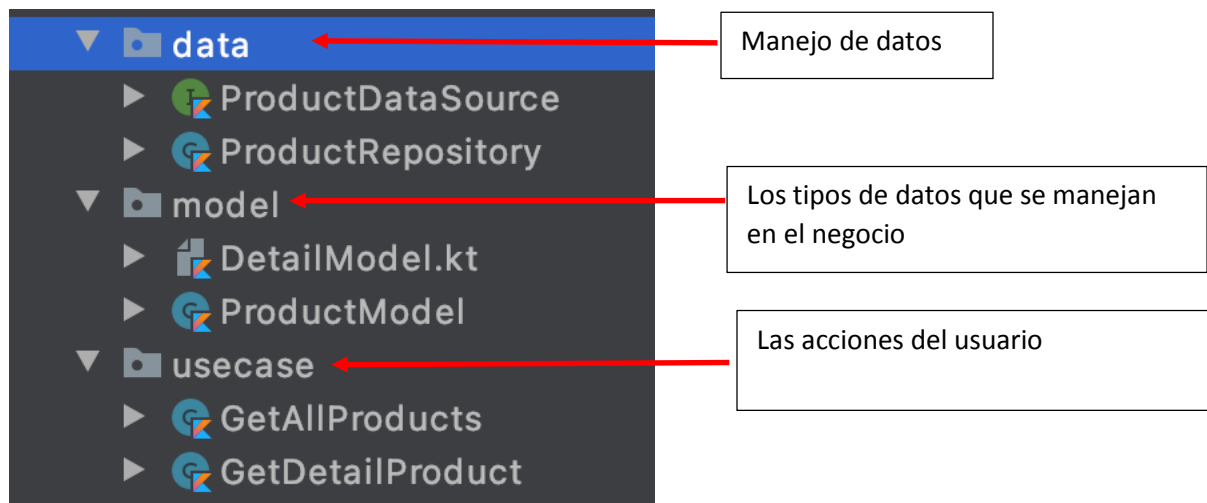
Se encarga de mostrar el listado de productos

Es la pantalla donde pone el texto a buscar, con validaciones básicas, como si hay o no internet o que este vacío al intentar una búsqueda

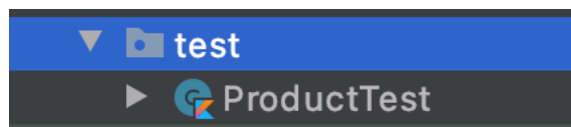
En **framework** encontramos lo relacionado al consumo del servicio (se usó retrofit para el consumo de las Api's)



En **core** tenemos lo relacionado al negocio:



Para desarrollo y pruebas agregue un paquete extra que es **test**



Con ello podemos probar diferentes escenarios que se pueden dar al ejecutar el challenge, por ejemplo, que el servicio no esté disponible o que la búsqueda regrese un listado vacío. Con ellos poder dar retro alimentación al usuario.

Para ejecutar estas pruebas solo hay que poner que se consuma el ProductTest en lugar de ProductService en ProductViewModel.kt y pasar un parámetro para verificar las diferentes respuestas.

```

22
23         //val repository = ProductRepository(ProductTest(0))           //Para pruebas y desarrollo
24                                     //0 regresa null
25                                     //1 regresa lista vacia
26                                     // otro numero regresa 3 productos
27
28         val repository = ProductRepository(ProductService()) //Productivo
29         val result: List<ProductModel>? = GetAllProducts(repository).invoke(query)

```

Lo mismo se puede realizar para el detalle en DetailViewModel.kt

```

24
25         //val repository = ProductRepository(ProductTest(0)) //Para pruebas y desarrollo
26         |                                     //0 regresa null
27                                     //otro valor regresa un detalle valido
28
29         val repository = ProductRepository(ProductService()) //Productivo
30

```

Gracias!!