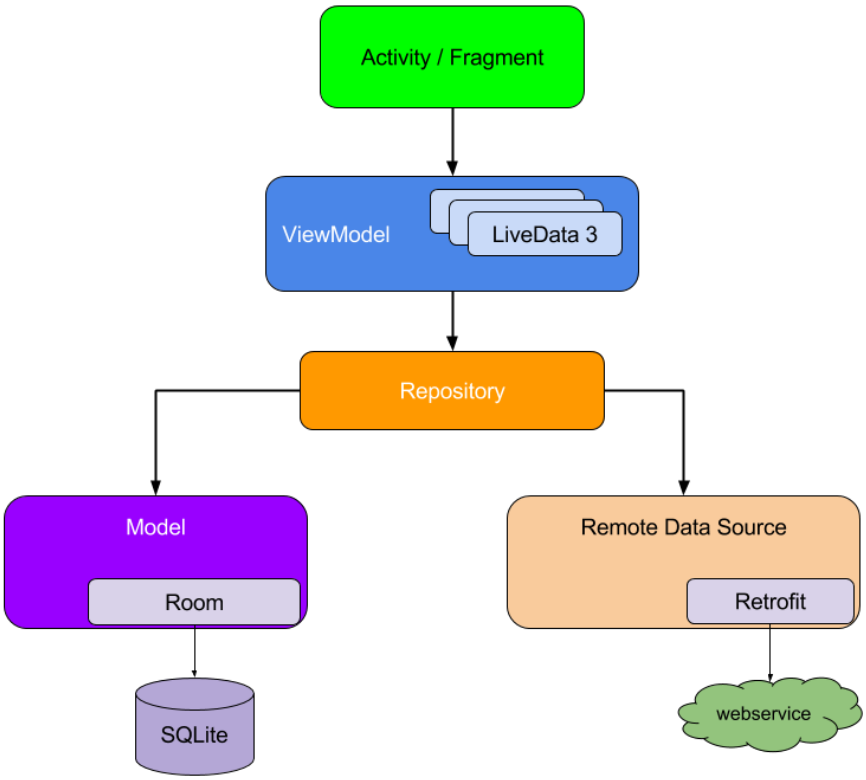
Arquitectura que se usará en la aplicación: MVVM

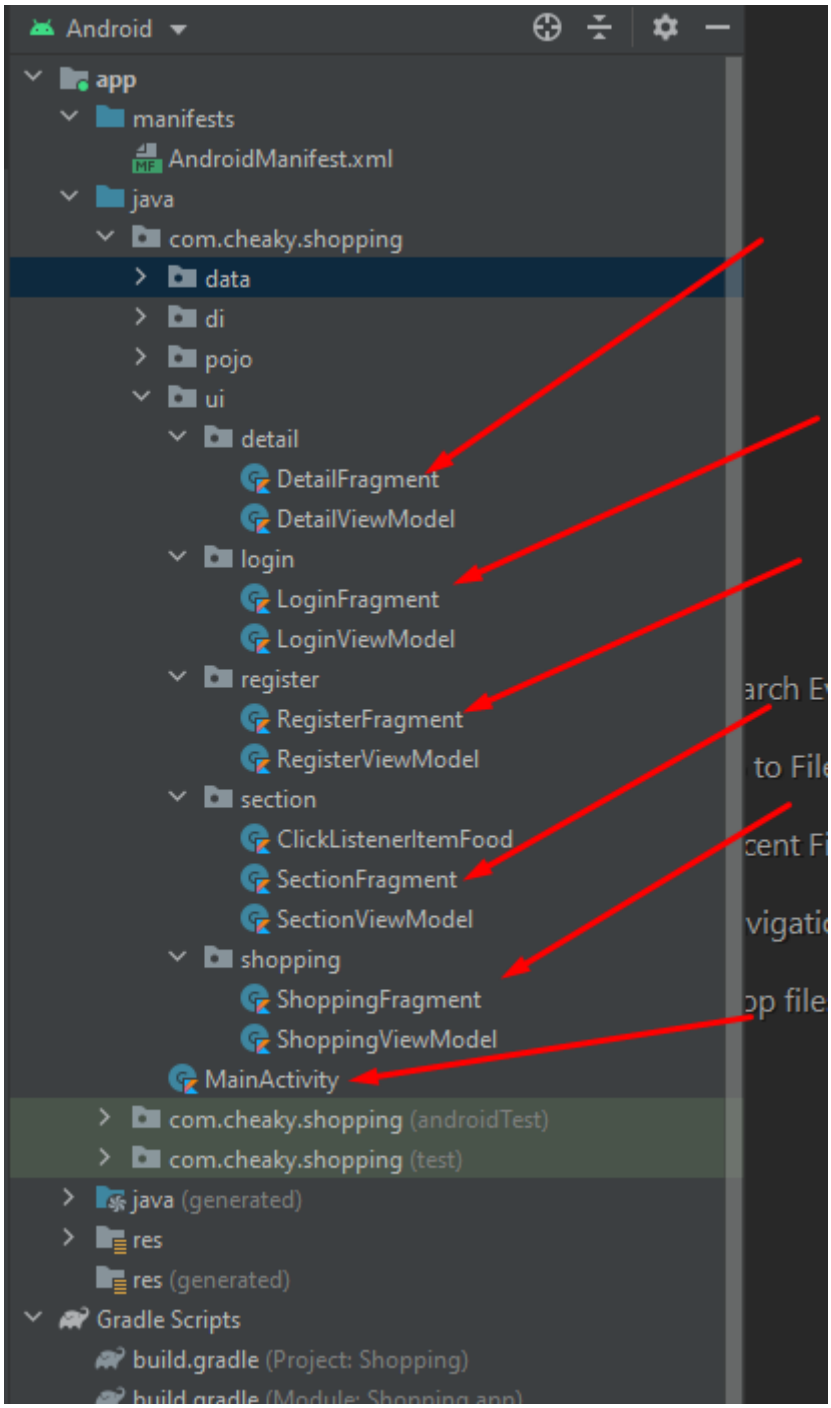


Esta arquitectura nos permitirá separar en módulos y asignar una única responsabilidad por capa, todos los módulos interactúan en sí, con la observación de que cada componente solo depende del componente que está un nivel más abajo, por ejemplo las actividades y los fragmentos solo dependen de un modelo de vista. El repositorio es la única clase que depende de otras clases, pero en nuestro caso al no trabajar con datos locales, solo dependeremos de la fuente de datos del backend remoto.

En la aplicación que se viene trabajando se cuenta con los diferentes componentes:

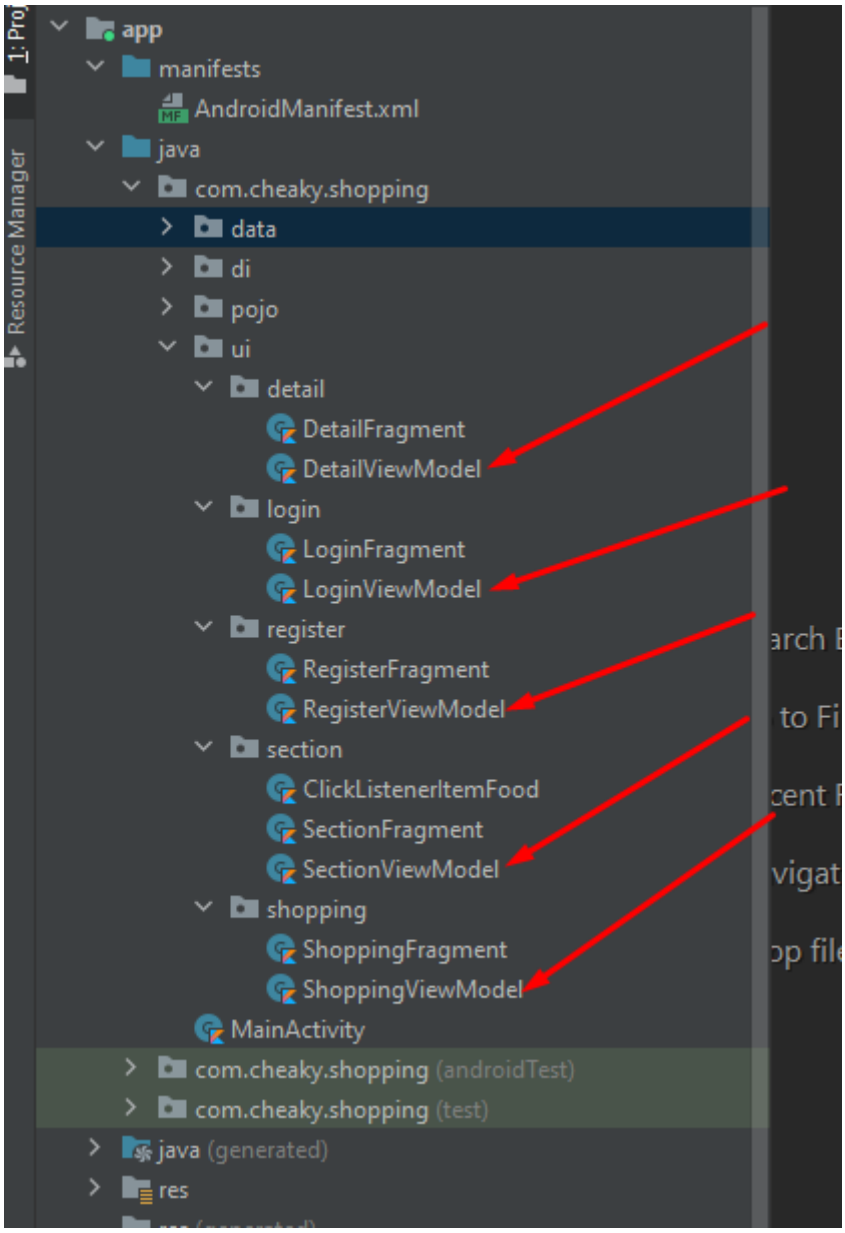
VISTAS: ACTIVITY Y FRAGMENT

También conocida como la capa de presentación, en palabras simples es donde definimos la interfaz de usuario y la interacción de éste con la app.



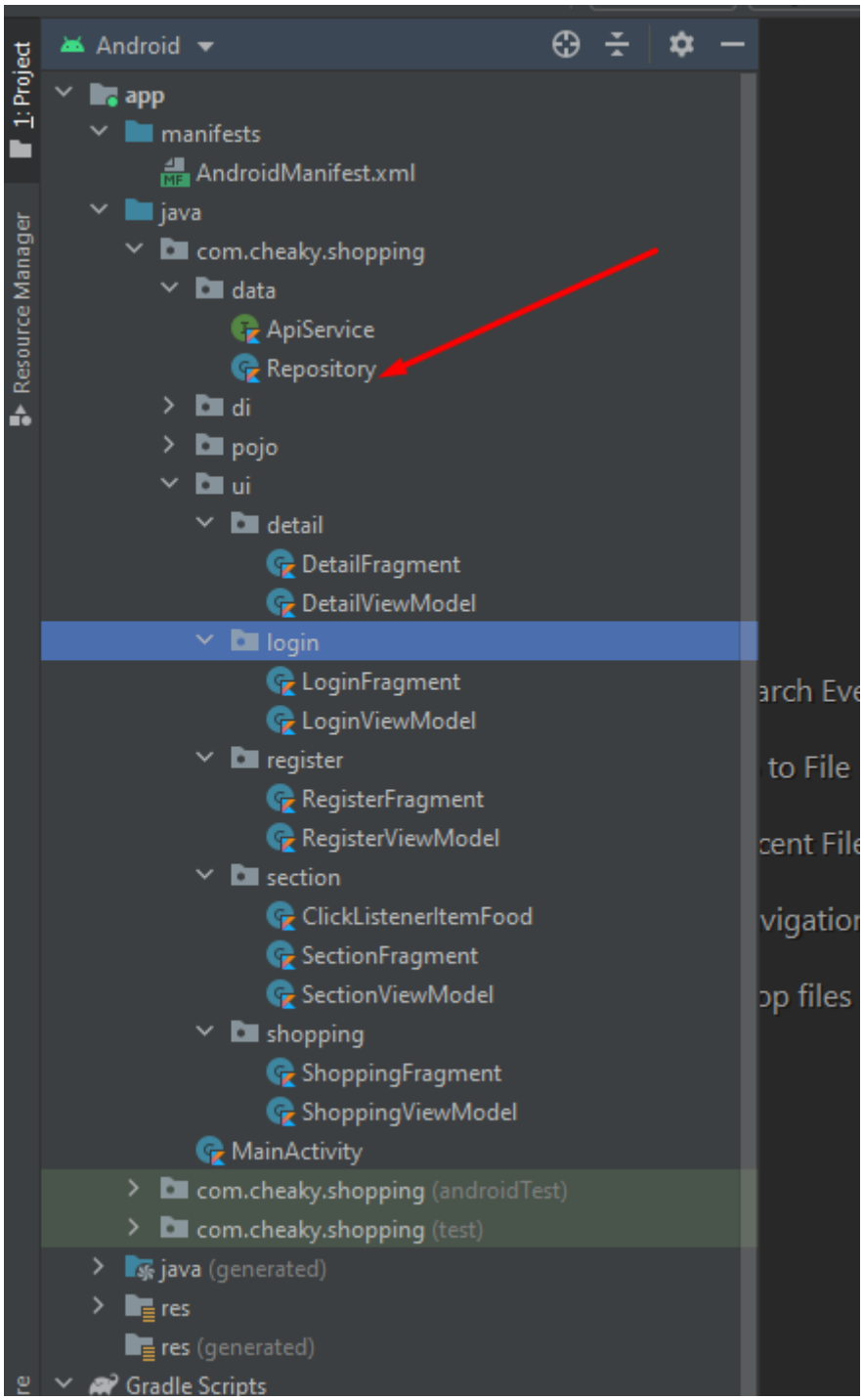
VIEWMODEL

Conocida también como la capa de negocio, es aquí donde irá nuestra lógica empresarial. Además de eso es aquí donde conservamos los datos después de un cambio de configuración como podría ser la rotación del dispositivo, con esto evitamos que los datos mostrados en la vista se pierdan o reinicien ya que el ciclo de vida de los viewmodels es distinta de al de los fragments o activities



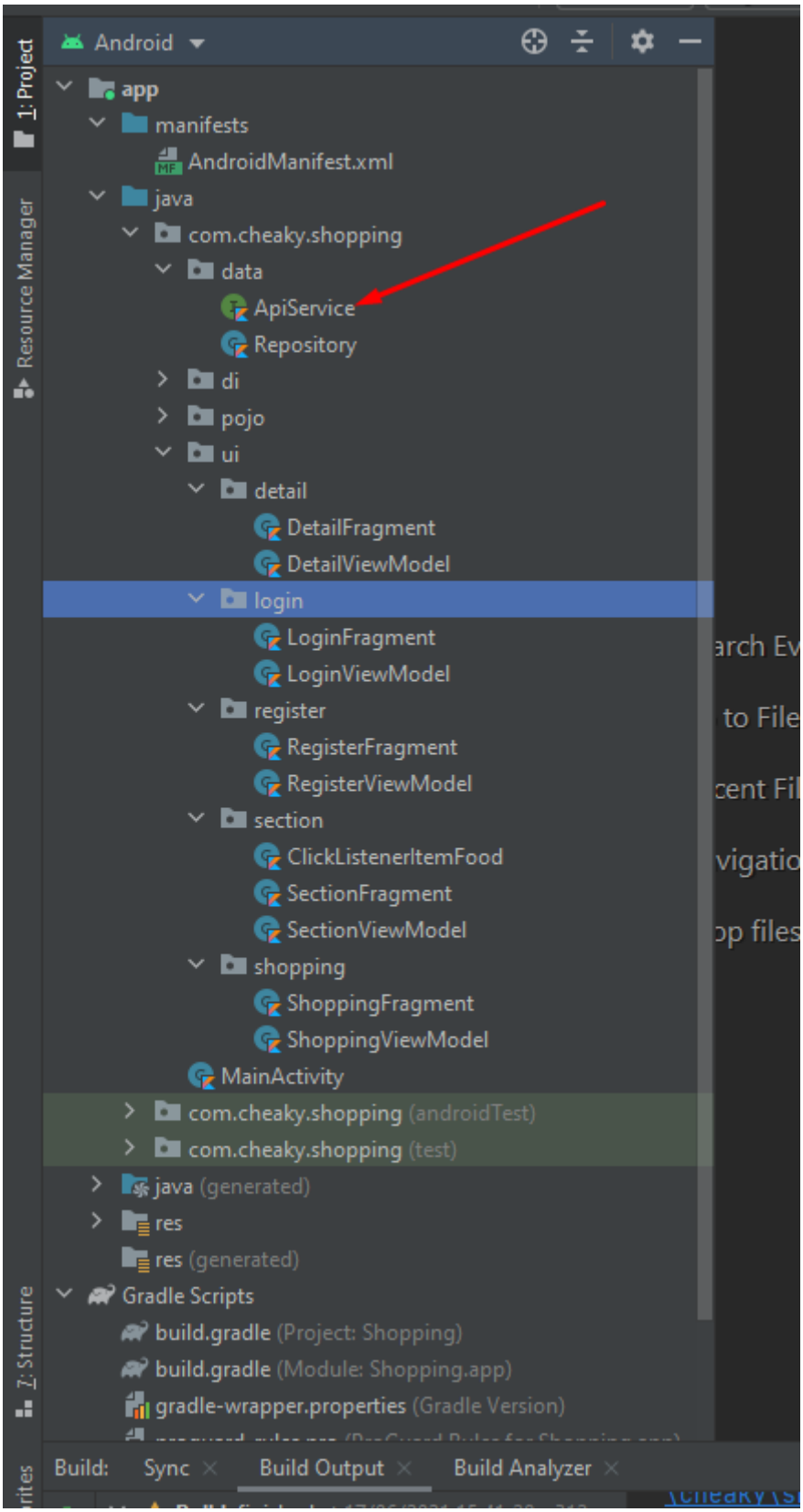
REPOSITORIO

Los módulos de repositorio manejan las operaciones de datos, nos proporcionan una API limpia para que el resto de la app pueda recuperar estos datos fácilmente. Saben de dónde obtener los datos y qué llamadas de API deben hacer cuando se actualizan los datos. Podemos considerar a los repositorios como mediadores entre diferentes fuentes de datos, como modelos persistentes, servicios web y memorias caché.



REMOTE DATA SOURCE

Como última capa tenemos la interfaz que nos permitirá comunicarnos con el web service (api rest), en ella podemos manejar diferentes solicitudes o métodos HTTP como GET, POST, UPDATE o DELETE



INYECCIÓN DE DEPENDENCIAS

Como extra necesitaremos un injector de dependencias para administrar las dependencias entre componentes, este patron nos permite hacer un escalamiento del código, ya que

proporcionan patrones claros para administrar dependencias sin duplicar el código ni aumentar la complejidad. Además, estos patrones nos permiten cambiar rápidamente entre las implementaciones de obtención de datos de producción y de prueba. Para este proyecto se usó la librería “DAGGER HILT” para poder administrar las dependencias