La arquitectura MVVM fue implementada en el proyecto FakeStore para estructurar y organizar el código de manera escalable y mantenible. Vamos a explicar cómo se implemento cada componente de esta arquitectura.

1. **Molde:**

El modelo se encarga de representar la capa de datos, las entidades y su origen.

**Implementación en FakeStore :**

**Entidades:**

* ProductsModel, CategoriesModel, UserModel y ShoppingCartModel representa la estructura de los datos que llegan desde la API.

**Origen de los datos**

* Se utiliza Retrofit para interactuar cola FakeStore API.
* FakeStoreRepository sirve como intermediario entre las capas de los datos y el ViewModel.

**Reponsabilidad del Model:**

* Manejar la lógica de obtención de datos desde la API.
* Transformar las respuestas de la API en objetos utilizables para la UI.
* Proveer datos al ViewModel a través de métodos como getProducts(), getUser() y getCartByUser()

1. **ViewModel:**

El ViewModel actúa como el intermedio entre el Modelo y la vista. Gestiona la lógica de negocio y gestiona el estado de la UI a través de flujos reactivos.

**Implementación en FakeStore:**

Cada módulo principal tiene su propio ViewModel:

* ProductsViewModel:
  + Gestiona la lógica para obtener los productos desde el repositorio.
  + Almacena el estado de los productos en un StateFlow que la vista puede observar para actualizar dinámicamente.
  + Contiene funciones como getProductsById() para obtener los detalles de un productos especifico.
* LoginViewModel:
  + Maneja la lógica de autenticación de usuario, como inicio de sesión y cierre de sesión.
  + Administrar el token de sesión de usuario y lo guarda en SharedPreferences para persistencia.
* CategoriesViewModel:
  + Gestiona la lógica de obtención de las categorías y los productos por categoría.
  + Expone un mapa que relaciona las categorías con sus productos, lo que facilita el acceso en las vistas.
* ShoppingCarViewModel:
  + Se encarga de obtener y exponer los productos del carrito de compras.

**Responsabilidades del ViewModel:**

* Gestionar los datos que se mostrarán en la vista, comunicándose con el repositorio.
* Exponer datos observables mediante StateFlow.

1. **View:**

La vista es la responsable de mostrar los datos al usuario y obtener sus interacciones.

**Implementación en FakeStore:**

Cada pantalla principal tiene su propia vista en la carpeta Views:

* ProductsView:
  + Observar el estado de los productos del ProductsViewModel usando collectAsState().
  + Muestra los productos en una lista usando LazyColumn o LazyVerticalGrid.
  + Navega a la pantalla de detalles del producto cuando el usuario selecciona un producto.
* CategoriesView:
  + Observa las categorías desde el CategoriesViewModel.
  + Muestra tarjetas de las categorías y permite navegar a los productos de una categoría especifica.
* LoginView:
  + Muestra el formulario de inicio de sesión.
  + Interactúa con LoginViewModel para autenticar al usuario y manejar los errores.
  + Redirige a las otras pantallas si el login es exitoso.

Responsabilidad de las vistas:

* Mostrar los datos proporcionados por el ViewModel.
* Capturar las interacciones del usuario y enviarlas al ViewModel para procesarlas.