Choices in Nectar Project

Judy Essam

August 4, 2025

1 the choice of adding two separate functions in cart

There are two functions for accessing a cart item, one that returns the flow of a cartItem and the other that just returns a direct cartItem object. The one for the Flow was added because when the user was incrementing and decrementing, the cart quantity did not reflect immediately. The reason I did not do this from the beginning – That is make the function return a flow from the beginning instead of have two separate ones – was that the initial function's purpose was to act as a "helper" in the other functions that manipulate the data in the data layer. This was bad design on my part, since I should have thought this one through more. However, I think that it generally would've added more complexity to use the Flow version of the function in the Repository.

Here I use the getCartItem (not flow) as a "helper" in the Repository:

```
override suspend fun incrementCartItem(product: product){
        val item = getCartItem(product.Id)
        if(item != null){
            val updated = item.copy(quantity = item.quantity + 1)
            updateCartItem(updated)
        }
        else{
            val new_item = cart(product = product, quantity = 1,
               productId = product.Id)
            insertCartItem(new_item)
        }
    }
```

Here I use the getCartItem (flow) to get the quantity of a cartItem in real time in the ViewModel

```
    val cartItemQuantity: StateFlow<Int> = _productId
        .filterNotNull()
        .flatMapLatest { id ->
            ObserveCartUseCase(id)
                .map { it?.quantity ?: 0 }
        }
        .stateIn(
            viewModelScope,
            SharingStarted.WhileSubscribed(5000),
            0
        )
```

2 the choice of the redundant ProductId in Cart

The reason I added a redundant ProductId column in the cart model was that it was simpler to include it in a column in itself since I shall be using it extensively to know whether a product was added to a cart or not, so that I can properly toggle between "adding" and "deleting" from a cart.

```
// in  CartViewModel Class
val cartItemIds = cart.map {
    items -> items.map {it.productId} }
    .stateIn(viewModelScope, SharingStarted.WhileSubscribed(5000),
       emptyList())

// e.g in CategoryScreen.kt

val cartItemIds by cartViewModel.cartItemIds.collectAsState()

items(products) { product ->
    ProductViewItem(
        title = product.productName,
        image = product.productImg,
        price = product.productPrice,
        details = product.productWeight,
        modifier = modifier.clickable(
            onClick = {navController.navigate("product/${product.Id}")
               }
        ),
        addCart = {cartViewModel.ToggleCartItem(product)},
         isInCart = cartItemIds.contains(product.Id)
    )
}
```

3 The choice of the fake search bar

In the HomeScreen and the Explore Screen, the search bars are not actual working search bars, but rather "fake ones" that when you press, gives the illusion of a search bar working. But in reality its not a real search bar but rather a button that navigates to the Search Screen, where an ACTUAL search bar resides.