Navegacion 3

```
// 1. Definir rutas (deben ser serializables)
sealed class Routes : NavKey {
    @Serializable data object Home : Routes()
    @Serializable data class Detalle(val id: String) : Routes()
}

// 2. Gestor de Navegación en MainActivity@Composable
fun GestionNavegacion() {
    val pila = rememberNavBackStack(elements = Routes.Home)
    NavDisplay(
        backStack = pila,
        onBack = { pila.removeLastOrNull() },
        entryProvider = { key ->
            when (key) {
                is Routes.Home -> NavEntry(key) {
                    PantHome(onNavega = { id -> pila.add(Routes.Detalle(id)) })
                }
                is Routes.Detalle -> NavEntry(key) {
                    PantDetalle(id = key.id, onBack = { pila.removeLastOrNull() })
                }
            }
        }
    )
}
```

Autenticación con Firebase

```
// Iniciar Sesión (Login)
auth.signInWithEmailAndPassword(email, password)
    .addOnSuccessListener { onLoginOk() }
    .addOnFailureListener { e -> Log.e("Firebase", "Error: ${e.message}") }

// Crear Usuario (Registro)
auth.createUserWithEmailAndPassword(email, password)
    .addOnSuccessListener { /* Navegar o avisar */ }
```

3. Operaciones CRUD en Firestore (MVVM)

```
class HomeViewModel : ViewModel() {
    private val db = Firebase.firestore
    private val productosColl = db.collection("productos")
    private val _productos = MutableStateFlow<List<Producto>>(emptyList())
    val productos: StateFlow<List<Producto>> = _productos
```

```
    init { getProductos() }

    // LEER: Escucha cambios en tiempo real
    private fun getProductos() {
        productosColl.addSnapshotListener { snapshot, _ ->
            if (snapshot != null) {
                _productos.value = snapshot.documents.mapNotNull { doc ->
                    doc.toObject(Producto::class.java)?.apply { id = doc.id }                }
            }
        }
    }

    // AÑADIR
    fun addProducto(nombre: String, precio: Double) {
        productosColl.add(Producto(nombre = nombre, precio = precio))
    }

    // ELIMINAR
    fun deleteProducto(id: String) {
        productosColl.document(id).delete()
    }

    // ACTUALIZAR (solo campos rellenos)
    fun updateProducto(id: String, nuevoNombre: String) {
        val datos = mutableMapOf<String, Any>()
        if (nuevoNombre.isNotBlank()) datos["nombre"] = nuevoNombre
        productosColl.document(id).update(datos)
    }
}
```

4. Componentes UI y Estados

```
@Composable
fun PantHome(viewModel: HomeViewModel = viewModel()) {
    val productos by viewModel.productos.collectAsState()
    var nombre by rememberSaveable { mutableStateOf("") }

    Column {
        TextField(value = nombre, onValueChange = { nombre = it }, label = { Text("Nombre") })
        Button(onClick = { viewModel.addProducto(nombre, 0.0) }) { Text("Añadir") }

        LazyColumn {
            items(productos) { prod ->
```

```
                Row(modifier = Modifier.clickable { /* Acción */ }) {
                    Text(prod.nombre, modifier = Modifier.weight(1f))
                    IconButton(onClick = { viewModel.deleteProducto(prod.id) }) {
                        Icon(Icons.Default.Delete, contentDescription = null)
                    }
                }
            }
        }
    }
}
```