15-12-2025

UNIVERSIDAD CENTRAL DEL ECUADOR

FACULTAD DE INGENIERÍA Y CIENCIAS APLICADAS

CARRERA DE COMPUTACIÓN

ASIGNATURA: Dispositivos Móviles

DOCENTE: Milton Giovanny Moncayo Unda

SEMESTRE: Octavo

TEMA: Tarea 07 - App Productos (MVC & Jetpack Compose)

INTEGRANTES:

- Carvajal Alexis
- Guamangallo Joel
- Minda Damian
- Moreno Billy
- Ortega David
- Palacios Jostyn

Documentación del Modelo de Navegación

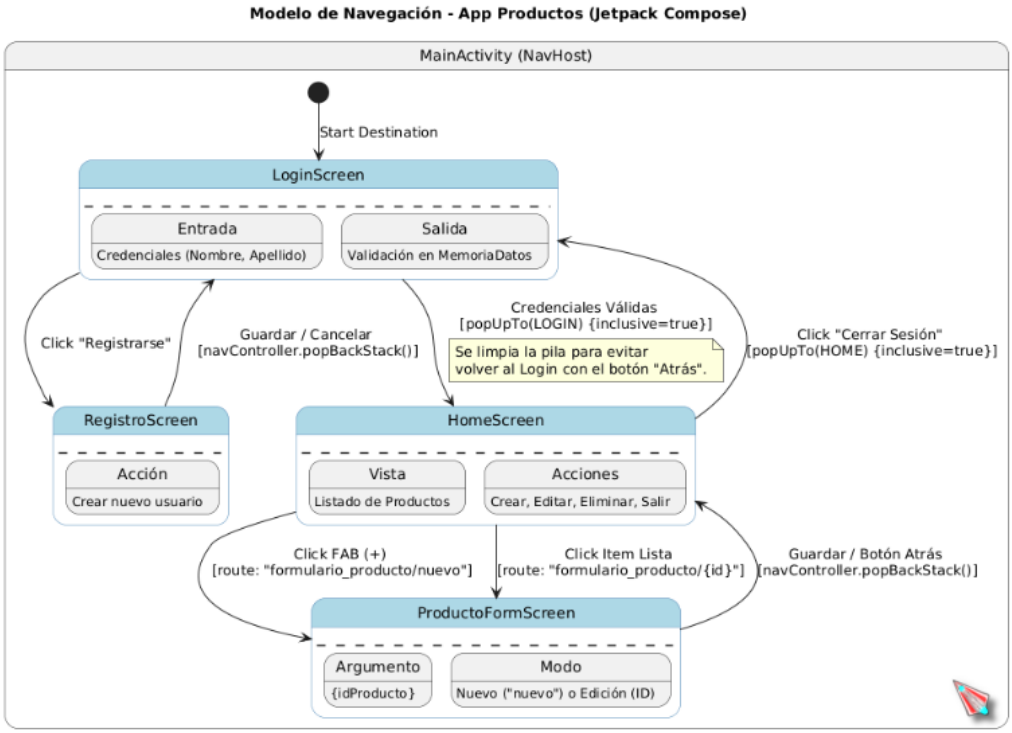
1. Arquitectura y Patrón de Diseño

Para el desarrollo de la aplicación "TechDrop", se ha implementado una arquitectura moderna basada en Jetpack Compose y el patrón de Navegación de Actividad Única (Single Activity Architecture). Aunque la interfaz es declarativa, el proyecto respeta la organización lógica del patrón Modelo-Vista-Controlador (MVC), adaptado a la siguiente estructura:

- MODELO (Model): Representa los datos y la lógica de negocio. Se implementó la persistencia mediante la clase MemoriaDatos (Singleton), garantizando que las listas de usuarios y productos se mantengan vivas durante el ciclo de vida de la aplicación sin requerir base de datos externa.
    - Archivos: Usuario.kt, Producto.kt, MemoriaDatos.kt.
- VISTA (View): Constituida por las pantallas (Screens) diseñadas con funciones @Composable. Estas vistas son reactivas y se actualizan automáticamente ante cambios en el modelo gracias al uso de mutableStateListOf, reflejando el estado actual de la interfaz gráfica.
    - Archivos: LoginScreen.kt, RegistroScreen.kt, HomeScreen.kt, ProductoFormScreen.kt.
- CONTROLADOR (Controller): El NavHost actúa como orquestador central, mientras que el NavController gestiona la navegación y la lógica de eventos (onClick). Este componente intercepta las acciones del usuario (login, guardar, eliminar) y decide qué operación realizar en el modelo o a qué pantalla navegar.
    - Archivos: NavGraph (definido en MainActivity), lógica de botones.

2. Mecanismo de Navegación (NavHost)

Específicamente para el flujo entre pantallas, se utiliza la librería Jetpack Navigation Compose. En este esquema, el NavHost actúa como un contenedor que intercambia dinámicamente los diferentes Composables según la ruta actual, eliminando la necesidad de crear múltiples Activities pesadas.



2.1. Mapa de Navegación (Grafo)

El flujo de la aplicación se define mediante las siguientes rutas:

1. **Login (startDestination):** Pantalla inicial de autenticación.
2. **Registro:** Permite crear nuevos usuarios y retorna al Login.
3. **Home:** Listado de productos (requiere autenticación exitosa).
4. **Formulario Producto:** Pantalla reutilizable que recibe argumentos.

Argumento idProducto: Si es "nuevo", muestra formulario vacío. Si es un ID existente, carga los datos para editar.

2.2. Lógica Detallada del Flujo

A. Flujo de Autenticación El módulo de seguridad gestiona el acceso inicial y protege las rutas privadas.

- Validación: En el Login, se busca linealmente en MemoriaDatos.listaUsuarios.
- Gestión del "Back Stack": Al ingresar al HOME, se ejecuta popUpTo(Rutas.LOGIN) { inclusive = true }. Esto elimina el historial de login para que el botón "Atrás" no devuelva al usuario a la pantalla de acceso.

B. Navegación en el Módulo de Productos La pantalla HomeScreen funciona como el Hub central.

- Rutas Dinámicas: Se optimizó el código utilizando una única ruta reutilizable:
- Crear: Se envía formulario_producto/nuevo.
- Editar: Se envía formulario_producto/{idProducto}.
- Cierre de Sesión: Al pulsar "Salir", se redirige al Login y se limpia la pila de navegación para cerrar la sesión efectivamente.

2.3. Implementación Técnica (Argumentos)

La lógica se centraliza en MainActivity.kt. A continuación, se evidencia cómo se captura el argumento idProducto:

```
// Definición de la ruta con argumento dinámico
composable(
    route = "formulario_producto/{idProducto}",
    arguments = listOf(navArgument("idProducto") { type = NavType.StringType })
) { backStackEntry ->
    val id = backStackEntry.arguments?.getString("idProducto")
    ProductoFormScreen(navController, id)
}
```

Esto permite reutilizar la misma interfaz gráfica para dos casos de uso distintos (Creación y Edición), optimizando el código.

2.4. Tabla de Especificación de Rutas

Para evitar errores de tipografía ("magic strings"), todas las rutas están centralizadas en un objeto Singleton Rutas.kt.

Ruta (Constante)	Patrón de URI	Argumentos	Descripción
`Rutas.LOGIN`	`"login"`	N/A	Pantalla de inicio de sesión.
`Rutas.REGISTRO`	`"registro"`	N/A	Formulario de registro de usuarios.
`Rutas.HOME`	`"home"`	N/A	Listado de productos (requiere auth).
`Rutas.FORM_PROD`	`"formulario_producto/{id}"`	`idProducto` (String)	Ruta híbrida. Si `id="nuevo"` muestra formulario vacío. Si es un ID existente, carga los datos para edición.

3. Evidencia de Funcionamiento (Capturas de Pantalla)

Figura 1. Pantalla de Login *(Pega aquí la captura del Login con tu logo TECHDROP)*
Descripción: Interfaz de inicio de sesión con validación de credenciales contra la lista en memoria.

Figura 2. Pantalla de Registro

Descripción: Formulario para añadir nuevos usuarios al Singleton MemoriaDatos.

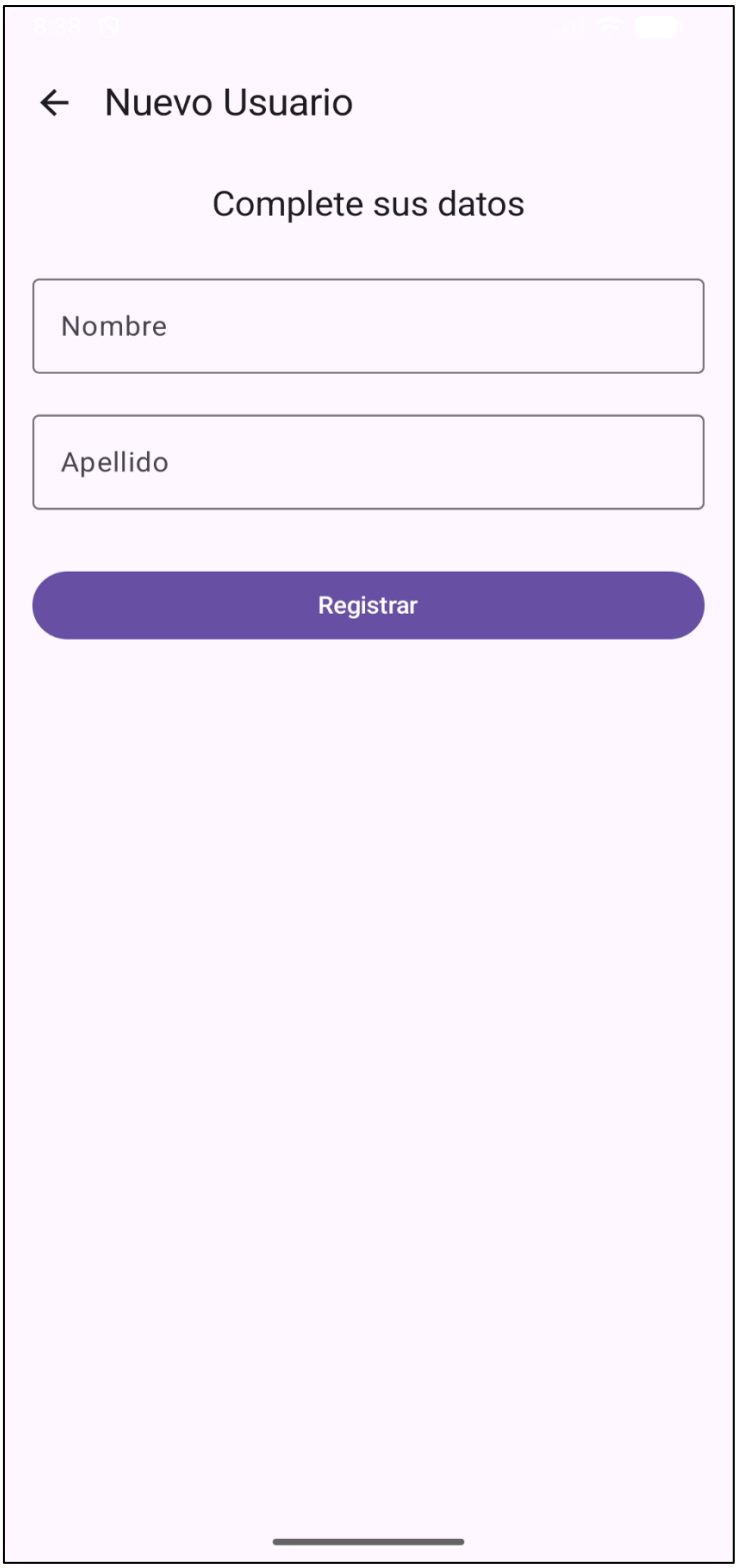


Figura 3. Pantalla Home (Lista de Productos)

Descripción: Listado reactivo de productos. Incluye botón flotante para agregar y opciones de editar/eliminar por ítem.

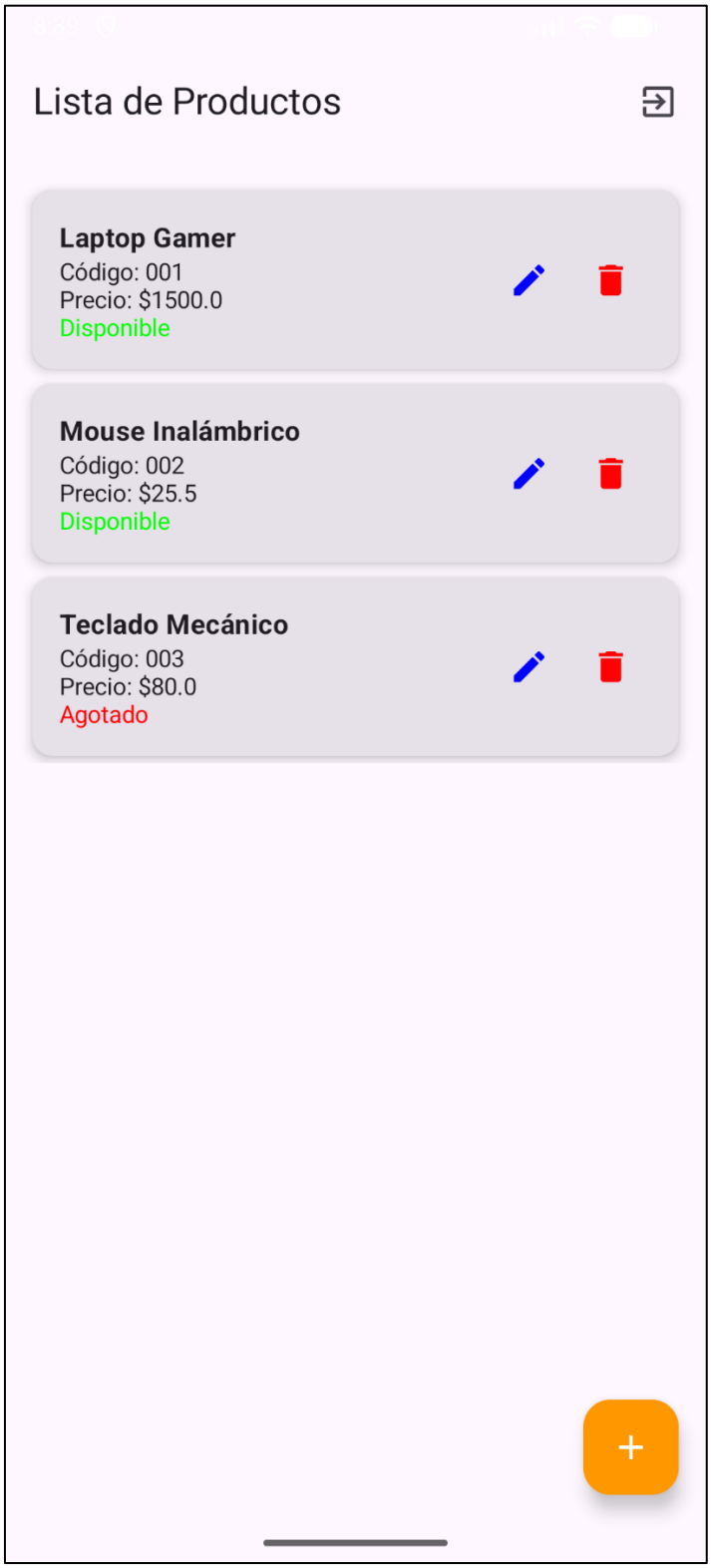


Figura 4. Confirmación de Eliminación

Descripción: Cuadro de diálogo AlertDialog para prevenir eliminaciones accidentales.

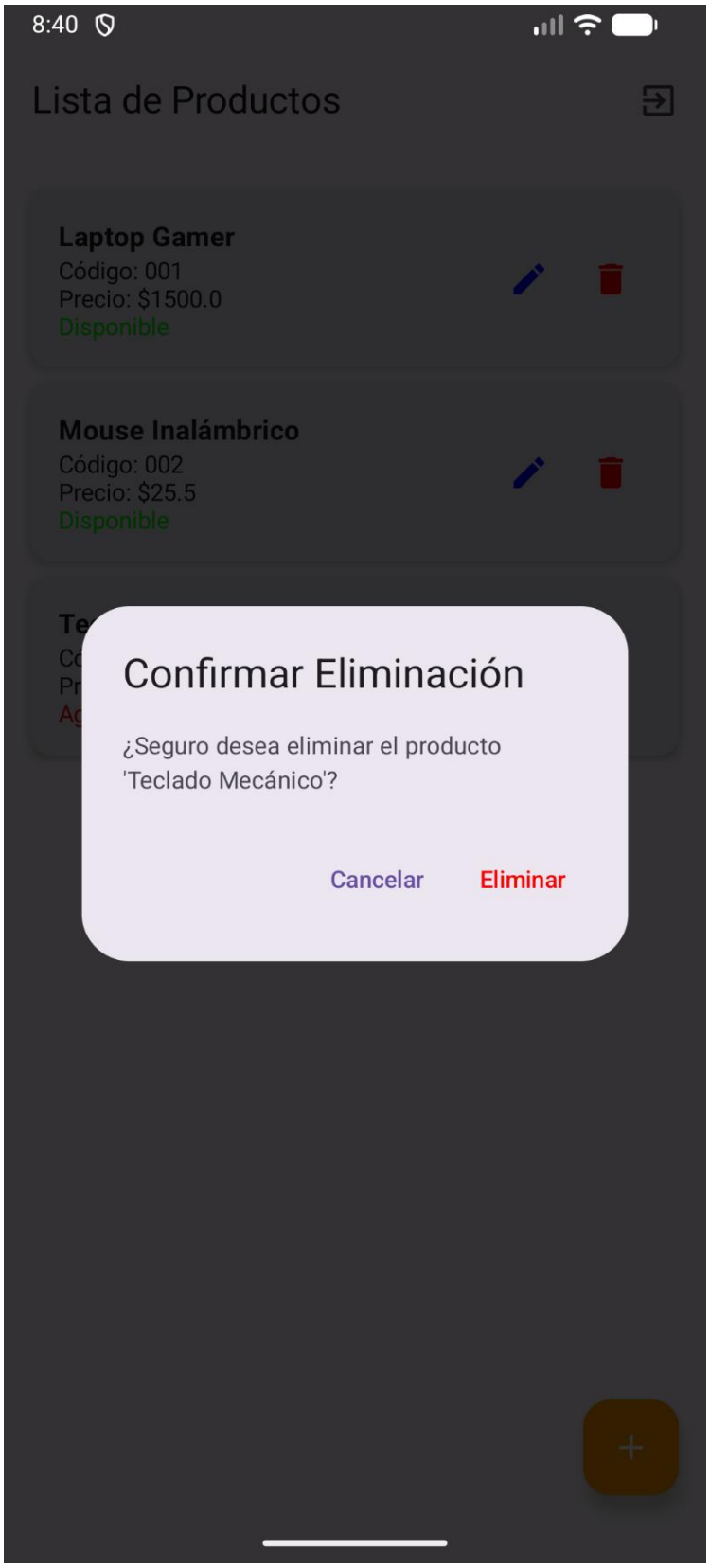


Figura 5. Formulario de Producto (Edición)

Descripción: La pantalla precarga los datos del producto seleccionado gracias al ID pasado por navegación.

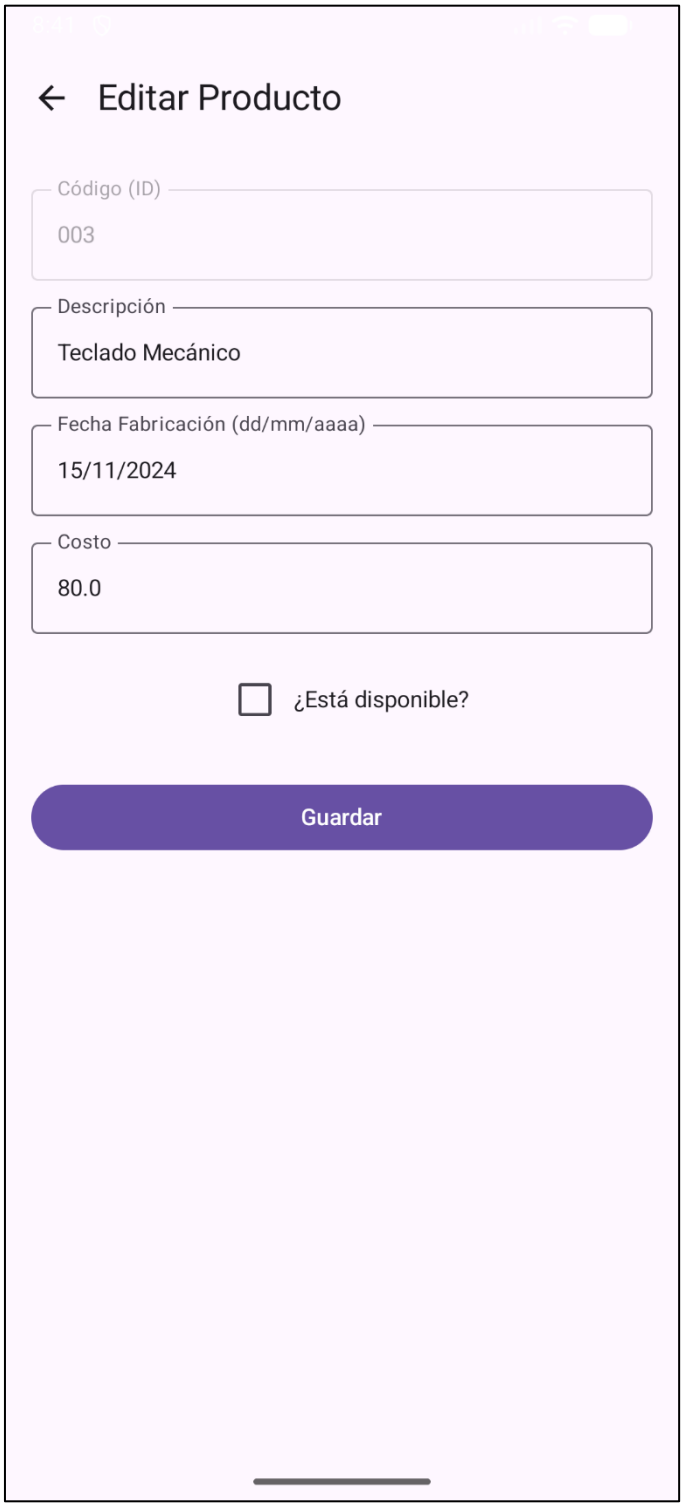


Figura 6. Pantalla de Nuevo Producto (Inserción)

Descripción: Esta pantalla permite al usuario ingresar los datos necesarios para registrar un nuevo producto en la aplicación, cumpliendo con la funcionalidad de inserción solicitada en los requerimientos. Los campos a completar corresponden a los datos del producto definidos previamente: código, descripción, fecha de fabricación, costo y una casilla de verificación para indicar su disponibilidad. Al hacer clic en el botón "Guardar", la información ingresada se almacena de forma persistente en la memoria de la aplicación.