TRABAJO FINAL

Desarrollar un Sistema **Offline first**, para una empresa dedicada a la venta de productos (Elija el tipo de productos)

La gestión debe estar orientada estrictamente al manejo del **Inventario** en base a: Compras, Almacenes y Ventas.

La empresa cuenta con: varias tiendas, varios almacenes y varios empleados

Objetivos del sistema en producción:

- Gestionar datos de: Productos, almacenes , tiendas y empleados
- Administrar y autenticar al encargado de cada tienda y almacén
- Alimentar datos de: Compras, Ventas y Transferencias de productos.
- Tener siempre actualizado: el inventario global, por tienda y por almacén,
- Reportes de Ventas y compras filtradas por tienda y fecha
- Reportes filtrados de transferencias de productos entre almacenes y tiendas
- Reporte de: Venta globales del día, por tienda y en general

No tomar en cuenta los módulos de: Planilla de sueldos, Activos fijos, Caja y banco, Control de asistencia del personal.

Stack: Dart, Flutter, Isar o Drift, Supabase

Duración del proyecto 4 semanas (fecha límite de entrega: 21 de Noviembre,: 24:00 horas)

Entregables:

Modelo relacional de las bases de datos (local y remoto)

apk, código fuente (cliente) y vídeo que exprese en extensión las funcionalidades integrales de los sistemas.

Web si tiene

Subir todos estos recursos a la nube, luego enviar un mensaje al email institucional: jtancara@ucb.edu.bo, indicando en el motivo del mensaje: **trabajo_final_movil_avanzado**_paterno_materno_nombres, en el cuerpo del mensaje incluir los enlaces para la descarga (asegurarse de que sean públicas no restringidas), además de los datos (login y password para probar)

RÚBRICA

Rúbrica de Evaluación: Sistema Offline-First

Requisito	Regular (Uso de GetX)	Buena (Uso de Riverpod)	Excelente (Uso de BLoC)
1. Funcionalidad (40%)			
1.1. Gestión de inventario	No autentica usuarios, permite CRUD de productos sin variantes y sin registro de stock por sucursal.	Autentica bien a los usuarios y permite CRUD de productos con variantes y registra entradas/salidas en una sola ubicación.	Autentica muy bien a los usuarios y permite CRUD de productos con variantes y registra entradas/salidas de forma detallada por sucursal y almacén, tomando en cuenta al autor de la transacción.
1.2. Gestión de ventas	Permite crear pedidos básicos y calcular el total.	Permite la creación de pedidos, el seguimiento de clientes y el historial de compras.	Permite la creación de pedidos, el seguimiento de clientes, la generación de recibos digitales y el cálculo de descuentos.
1.3. Gestión de sucursales	No diferencia entre sucursales o almacenes.	La aplicación permite seleccionar la sucursal, pero no muestra el stock específico de cada una.	La aplicación permite seleccionar sucursal y almacén, mostrando el stock disponible en cada ubicación.
1.4. Experiencia de usuario	Interfaz de usuario funcional pero con una estética básica.	La interfaz es intuitiva y sigue parcialmente las guías de Material Design.	Interfaz de usuario intuitiva, con diseño profesional que sigue las guías de Material Design, animaciones fluidas y transiciones lógicas.
2. Arquitectura Offline-First (30%)			
2.1. Persistencia local	Los datos se almacenan de forma local, pero el esquema de la base de datos es simple y no está optimizado.	El esquema de la base de datos local (Isar/Drift) está bien diseñado para la funcionalidad de la app.	El esquema de la base de datos local es robusto, optimizado y refleja una comprensión profunda de las necesidades del sistema.

Requisito	Regular (Uso de GetX)	Buena (Uso de Riverpod)	Excelente (Uso de BLoC)
2.2. Manejo de conectividad	La aplicación detecta si hay conexión, pero no hay una retroalimentación clara en la interfaz de usuario.	La aplicación detecta el estado de la conexión y lo muestra con un banner o icono.	La aplicación no solo detecta el estado de la conexión, sino que adapta activamente la interfaz para indicar qué acciones se pueden realizar.
2.3. Sincronización	Los datos se envían al servidor cuando hay conexión, sin una cola de tareas.	Se implementa una cola de tareas para las operaciones offline, pero la lógica de resolución de conflictos es simple o inexistente.	Se implementa una cola de tareas robusta y una lógica de resolución de conflictos bien definida (ej. uso de **timestamps** para determinar la última versión).
3. Implementación Técnica (30%)			
3.1. Arquitectura de Estado	El código está acoplado al usar GetX, con una gestión de estado no clara y con posibles fugas de memoria.	El código usa Riverpod, con una gestión de estado bien separada en Providers, haciendo la app modular y fácil de testear.	El código usa BLoC, implementando una arquitectura limpia y sólida con una clara separación de la lógica de negocio, haciendo la app escalable y mantenible.
3.2. Uso de Supabase	Se usa Supabase para funciones básicas (autenticación y base de datos).	Se usa Supabase de forma efectiva para la base de datos y la autenticación, con consultas optimizadas.	Se usan las **Políticas de Seguridad a Nivel de Fila (RLS)** y las capacidades en tiempo real de Supabase para la sincronización automática de cambios del servidor.

3.3. Calidad del código	Código legible pero con poca documentación y dependencias acopladas, producto poco amigable	Código limpio, modular y con comentarios esenciales que explican las partes complejas. Producto amigable con tema rígido.	Código bien documentado y una arquitectura de proyecto que facilita su escalabilidad. Producto altamente amigable y tema versátil configurable.