**APLICACIÓN CAJA MERCADILLOS - DOCUMENTACIÓN COMPLETA Y DETALLADA**

**1. INTRODUCCIÓN Y CONTEXTO DEL PROYECTO**

**1.1 Origen del Proyecto**

La aplicación "Caja Mercadillos" nace de una necesidad real identificada por un desarrollador que tiene experiencia directa vendiendo en mercadillos ambulantes de productos artesanos. Durante nuestra conversación, el desarrollador explicó que esta experiencia práctica le ha permitido identificar las carencias específicas de las soluciones existentes en el mercado y las particularidades únicas de este tipo de negocio.

El proyecto surge con la premisa de crear una herramienta específicamente diseñada para el control de caja de puestos ambulantes, reconociendo que las aplicaciones generalistas de punto de venta no se adaptan adecuadamente a las necesidades específicas de los mercadillos artesanos.

**1.2 Filosofía de Desarrollo**

Durante nuestra conversación se estableció claramente que el desarrollo se basa en la filosofía de "resolver problemas reales con soluciones prácticas". Cada funcionalidad ha sido pensada para abordar situaciones específicas que se presentan en la operación diaria de un puesto de mercadillo, evitando complejidades innecesarias y centrándose en la usabilidad práctica.

El desarrollador enfatizó la importancia de tener una "visión global" del proyecto antes de implementar cualquier funcionalidad, para evitar el problema común de "arreglar una cosa y estropear otra" que había experimentado con otras inteligencias artificiales.

**1.3 Objetivos Principales**

Los objetivos del proyecto, tal como se definieron en nuestra conversación, son:

* Crear una aplicación de control de caja específicamente adaptada a puestos ambulantes
* Implementar un sistema que funcione de manera confiable tanto con conexión a internet como sin ella
* Proporcionar herramientas de gestión que se adapten al flujo real de trabajo de un mercadillo
* Ofrecer un modelo de negocio escalable que permita acceso gratuito con opción de actualización premium

**2. MODELO DE NEGOCIO Y ESTRATEGIA COMERCIAL**

**2.1 Estructura de Versiones**

**2.1.1 Versión FREE (Gratuita)**

Durante nuestra conversación se definió que la versión gratuita sería "monopuesto", lo que significa que solo permite un dispositivo activo simultáneamente. Esta estrategia fue explicada detalladamente:

**Características principales de la versión FREE:**

* Sistema monopuesto con control estricto de dispositivos
* Funcionalidades básicas completas de control de caja
* Gestión de categorías y artículos sin restricciones
* Sistema de ventas estándar con todas las formas de pago
* Sincronización básica con la nube
* Acceso a todas las funcionalidades core sin limitaciones artificiales

**Mecánica del sistema monopuesto:** El desarrollador explicó que quería implementar un sistema similar al funcionamiento de WhatsApp. Cuando un usuario inicia sesión en un dispositivo nuevo:

1. Los datos se descargan automáticamente al nuevo dispositivo
2. El dispositivo anterior queda automáticamente inoperativo
3. No hay uso simultáneo en múltiples dispositivos
4. El usuario puede cambiar de dispositivo cuando lo necesite (por ejemplo, si se le rompe el teléfono)

**Justificación comercial:** Esta estrategia permite ofrecer una funcionalidad completa y útil en la versión gratuita mientras incentiva la actualización a premium para usuarios que necesiten trabajar con múltiples dispositivos. Es especialmente apropiada para vendedores ambulantes individuales que normalmente trabajan con un solo dispositivo.

**2.1.2 Versión PREMIUM (Pago único)**

La versión premium se diseñó con un modelo de pago único (no suscripción) para hacerla más atractiva a los vendedores ambulantes, quienes según la experiencia del desarrollador, suelen tener ingresos variables y prefieren evitar gastos recurrentes.

**Características adicionales de la versión PREMIUM:**

* Sistema multipuesto sin restricciones de dispositivos
* Control avanzado de stock con configuración granular por artículo
* Control de precios de coste para cálculo de márgenes de beneficio
* Sistema de reportes y análisis avanzados
* Funcionalidades de backup y exportación de datos
* Gestión de usuarios con diferentes roles y permisos (para negocios más grandes)
* Funcionalidades adicionales como fotos de artículos y códigos de barras

**2.2 Sistema de Autenticación y Control de Dispositivos**

**2.2.1 Autenticación con Google**

Durante nuestra conversación se estableció que el sistema utilizaría Google OAuth para la autenticación, integrado con Firebase Authentication. Esta decisión se basó en:

* Simplicidad para el usuario (no necesita crear otra cuenta)
* Robustez del sistema de Google
* Integración natural con Firebase
* Reducción de la fricción en el proceso de registro

**2.2.2 Control de Dispositivos Monopuesto**

El sistema de control monopuesto fue explicado en detalle durante nuestra conversación:

**Mecánica técnica:**

1. Firebase Authentication maneja el login con Google
2. La tabla de configuración almacena el identificador del dispositivo activo
3. Cuando la aplicación detecta un login desde un dispositivo diferente al registrado:
   * Compara el ID del dispositivo actual con el almacenado en la configuración
   * Si son diferentes, descarga todos los datos al nuevo dispositivo
   * Actualiza el identificador de dispositivo en la configuración
   * El dispositivo anterior pierde acceso (queda inoperativo)

**Implementación del control:** El desarrollador explicó que el dispositivo anterior se vuelve inoperativo porque al intentar sincronizar datos, detecta que ya no es el dispositivo autorizado. Esto efectivamente "desactiva" el dispositivo sin necesidad de notificaciones push complejas.

**3. ARQUITECTURA TÉCNICA Y GESTIÓN DE DATOS**

**3.1 Sistema Offline-First**

**3.1.1 Filosofía de la Arquitectura**

Una de las decisiones más importantes discutidas en nuestra conversación fue implementar una arquitectura "offline-first". El desarrollador explicó que esto era crucial para los mercadillos, donde la conexión a internet puede ser intermitente o inexistente.

**Principios fundamentales:**

* La aplicación debe funcionar completamente sin conexión a internet
* Todas las operaciones deben ser instantáneas para el usuario
* Los datos nunca se pierden, independientemente de problemas de conectividad
* La sincronización debe ser transparente e invisible para el usuario

**3.1.2 Flujo de Datos Detallado**

El flujo de datos fue explicado meticulosamente durante nuestra conversación:

**Proceso de escritura:**

1. **Operación local primaria**: Toda operación (crear categoría, registrar venta, etc.) se guarda inmediatamente en Room (base de datos local)
2. **Respuesta inmediata**: La UI se actualiza instantáneamente mostrando los cambios
3. **Verificación de conectividad**: El sistema verifica si hay conexión a internet
4. **Sincronización condicional**:
   * Si hay conexión: Los datos se suben inmediatamente a Firebase
   * Si no hay conexión: Los datos quedan marcados como "pendientes de sincronizar"

**Proceso de sincronización automática:**

1. **Detección de conectividad**: La aplicación monitorea constantemente el estado de la conexión
2. **Sincronización automática**: Cuando se detecta que se ha recuperado la conexión, automáticamente sincroniza todos los datos pendientes
3. **Transparencia total**: El usuario nunca sabe ni necesita saber dónde están almacenados sus datos

**3.1.3 Ventajas de esta Arquitectura**

Durante nuestra conversación se identificaron múltiples ventajas:

* **Fiabilidad absoluta**: La aplicación siempre funciona, con o sin internet
* **Performance óptimo**: Las operaciones son instantáneas al usar la base de datos local
* **Robustez ante fallos**: No se pierden datos por problemas de conectividad
* **Experiencia de usuario superior**: No hay esperas ni interrupciones por problemas de red
* **Adaptación al contexto**: Perfecto para mercadillos donde la conectividad es variable

**3.2 Tecnologías y Patrones Arquitectónicos**

**3.2.1 Stack Tecnológico**

Las tecnologías fueron seleccionadas cuidadosamente durante nuestra conversación:

**Tecnologías principales:**

* **Kotlin**: Lenguaje de programación moderno y robusto para Android
* **Jetpack Compose**: Framework de UI declarativa y reactiva para interfaces modernas
* **Room**: ORM para gestión de base de datos local SQLite
* **Firebase Firestore**: Base de datos NoSQL en la nube para sincronización
* **Firebase Authentication**: Sistema de autenticación robusto
* **Coroutines + Flow**: Para programación asíncrona y manejo de datos reactivos

**Librerías adicionales:**

* **Skydoves ColorPicker**: Para selector de colores profesional en categorías
* **Material Design 3**: Para componentes UI consistentes y modernos

**3.2.2 Patrones Arquitectónicos Implementados**

Durante nuestra conversación se establecieron los patrones a seguir:

**MVVM (Model-View-ViewModel):**

* **Model**: Entidades de datos y repositorios
* **View**: Composables de UI reactivos
* **ViewModel**: Lógica de presentación y gestión de estado

**Repository Pattern:**

* Abstrae las fuentes de datos (Room local + Firebase remoto)
* Implementa la lógica offline-first
* Proporciona interfaces limpias a los ViewModels
* Maneja automáticamente la sincronización

**Clean Architecture:**

* Separación clara de responsabilidades en capas
* Dependencias que apuntan hacia adentro
* Reglas de negocio independientes de frameworks

**4. FUNCIONALIDADES DETALLADAS**

**4.1 Gestión de Mercadillos**

**4.1.1 Concepto y Importancia**

Durante nuestra conversación, el desarrollador explicó en detalle cómo funciona un mercadillo desde su experiencia real. Un mercadillo es un evento comercial específico que tiene características únicas:

* Fecha y horario específicos
* Ubicación determinada
* Organizador particular
* Condiciones logísticas específicas (precio del puesto, si incluye mesa, carpa, electricidad)

El desarrollador enfatizó que la gestión de mercadillos es el núcleo de la aplicación, ya que todo el control de caja se organiza alrededor de estos eventos comerciales.

**4.1.2 Información de un Mercadillo**

Se definieron detalladamente los datos que debe almacenar cada mercadillo:

**Datos obligatorios:**

* **Fecha y hora**: Cuándo se celebra el mercadillo
* **Ubicación**: Dirección exacta o descripción del lugar
* **Organizador**: Quién organiza el evento (importante para contacto y pagos)
* **Hora de inicio**: Momento oficial de apertura

**Datos opcionales:**

* **Nombre del mercadillo**: Identificación personalizada ("Mercadillo de San Juan", "Feria Artesana de Primavera")
* **Hora de finalización**: Para control de duración del evento
* **Información logística**: Precio del puesto, si incluye electricidad, si requiere carpa o mesa
* **Notas personales**: Observaciones sobre acceso, parking, público objetivo, etc.

**4.1.3 Vista de Calendario**

El desarrollador explicó la necesidad de tener una vista de calendario que muestre todos los mercadillos programados. Se definió un sistema de códigos de colores para identificar rápidamente el estado de cada mercadillo:

**Códigos de colores propuestos:**

* **Azul**: Mercadillo programado (futuro, aún no iniciado)
* **Verde**: Mercadillo en curso (actualmente activo)
* **Verde oscuro**: Mercadillo finalizado con beneficios
* **Rojo**: Mercadillo finalizado con pérdidas
* **Gris**: Mercadillo cancelado
* **Amarillo**: Mercadillo con arqueo pendiente de cerrar

**4.1.4 Estados de Mercadillo**

Durante nuestra conversación se identificaron los diferentes estados que puede tener un mercadillo:

1. **Programado**: Solo tiene la información básica registrada, sin saldo asignado
2. **Con saldo asignado**: Ya tiene definido el saldo inicial, listo para comenzar la actividad
3. **En curso**: Está actualmente activo, registrando ventas y gastos
4. **Finalizado**: Completado con arqueo de caja cerrado
5. **Cancelado**: No se realizó por cualquier motivo
6. **Pendiente de arqueo**: La actividad comercial terminó pero falta cerrar contablemente

**4.2 Gestión de Saldos Entre Mercadillos**

**4.2.1 Filosofía del Sistema de Saldos**

El desarrollador explicó detalladamente cómo funciona el manejo de efectivo entre mercadillos en la realidad práctica. Normalmente, el dinero que queda en caja al final de un mercadillo se convierte en el saldo inicial del siguiente mercadillo. Sin embargo, entre mercadillos pueden ocurrir movimientos de dinero que afectan este traspaso.

**4.2.2 Flujo Normal de Saldos**

Se estableció el flujo típico:

1. **Mercadillo A**: Comienza con saldo inicial X euros
2. **Durante el día**: Se realizan ventas (ingresos) y gastos
3. **Al cerrar**: Se calcula el saldo final Y euros
4. **Mercadillo B**: Toma como saldo inicial los Y euros del mercadillo anterior

**4.2.3 Proceso de Cierre y Asignación de Saldo**

El desarrollador describió en detalle cómo debería funcionar el proceso de cierre:

**Cuando se completa un mercadillo:**

1. **Cálculo automático**: La aplicación calcula el saldo final en efectivo basado en el arqueo
2. **Pantalla de gestión**: Se presenta una interfaz para gestionar el saldo
3. **Identificación del siguiente mercadillo**: Si hay un mercadillo próximo programado, se muestra automáticamente
4. **Opciones de ajuste**:
   * **Campo "Retirar dinero"**: Cantidad que el vendedor se lleva (beneficios, gastos personales)
   * **Campo "Añadir dinero"**: Dinero adicional que se mete en caja (cambio extra, billetes pequeños)
5. **Cálculo final**: Saldo inicial próximo mercadillo = Saldo final actual - Dinero retirado + Dinero añadido

**4.2.4 Casos Especiales en la Gestión de Saldos**

**Cuando no hay mercadillo siguiente programado:** El desarrollador explicó que a veces no se sabe cuándo será el próximo mercadillo. En estos casos:

* El mercadillo actual se puede cerrar completamente
* El saldo final queda marcado como "pendiente de asignar"
* Cuando se programe un futuro mercadillo, la aplicación preguntará si se quiere usar este saldo pendiente
* Se mantienen las mismas opciones de retirar/añadir dinero

**Reasignación de saldos:** Se discutió la posibilidad de permitir reasignar saldos si se comete un error o cambian los planes.

**4.2.5 Gestión de Mercadillos Cancelados**

El desarrollador identificó dos escenarios diferentes para mercadillos cancelados:

**Cancelación simple:**

* Mercadillo que solo tiene los datos básicos (fecha, lugar, organizador)
* No tiene saldo inicial asignado
* **Acción**: Simplemente marcar como "Cancelado"

**Cancelación con saldo asignado:**

* Mercadillo que ya tenía saldo inicial asignado pero se cancela después
* **Proceso de cierre especial**:
  1. La aplicación detecta que no hubo actividad comercial (sin ventas ni gastos)
  2. Pregunta al usuario: "¿Este mercadillo ha sido cancelado?"
  3. Si se confirma, marca el mercadillo como "Cancelado"
  4. Establece automáticamente: saldo final = saldo inicial
  5. Ofrece las mismas opciones de retirar/añadir dinero
  6. Pregunta si se quiere asignar este saldo a otro mercadillo o dejarlo pendiente

**4.3 Sistema de Ventas**

**4.3.1 Filosofía de Flexibilidad en Ventas**

Durante nuestra conversación, el desarrollador enfatizó que el sistema de ventas debe ser extremadamente flexible para adaptarse a la realidad de un mercadillo, donde se pueden vender tanto productos habituales como artículos únicos o personalizados.

**4.3.2 Tipos de Artículos en Ventas**

Se definieron tres modalidades para registrar ventas:

**Artículos predefinidos:**

* Productos que están registrados en la base de datos de artículos
* Tienen precio establecido, categoría asignada, y potencialmente control de stock
* Se seleccionan de una lista o catálogo
* **Ventaja**: Agilizan enormemente el proceso de venta para productos habituales
* **Ejemplo**: "Camiseta básica - 15€"

**Artículos manuales:**

* Productos que se introducen al momento de la venta
* Solo requieren descripción y precio
* **Uso típico**: Artículos únicos, personalizados, servicios, o productos excepcionales
* No afectan inventarios ni estadísticas de productos regulares
* **Ejemplo**: "Personalización bordado - 5€"

**Tickets híbridos:**

* Combinación de artículos predefinidos y manuales en una sola venta
* **Máxima flexibilidad** para situaciones reales del mercadillo
* **Ejemplo práctico**: "Camiseta básica (predefinida) 15€ + Personalización bordado (manual) 5€ = Total 20€"

**4.3.3 Proceso de Venta Detallado**

**Fase 1: Construcción del ticket**

1. **Selección de artículos predefinidos**: El vendedor busca y selecciona productos del catálogo
2. **Adición de artículos manuales**: Introduce descripción y precio de productos no catalogados
3. **Gestión de cantidades**: Puede modificar las cantidades de cualquier artículo
4. **Visualización en tiempo real**: Ve el total actualizado constantemente
5. **Modificaciones**: Puede eliminar o modificar artículos antes de confirmar

**Fase 2: Confirmación del ticket**

1. **Resumen completo**: Se muestra una lista detallada de todos los artículos
2. **Total claramente visible**: El importe total a cobrar se destaca
3. **Última oportunidad**: El vendedor puede modificar antes de proceder al pago
4. **Confirmación**: Al confirmar, se procede a la selección de forma de pago

**Fase 3: Selección de forma de pago** El desarrollador explicó que el sistema debe soportar las tres formas de pago más comunes en mercadillos:

**Bizum:**

* Proceso directo sin cálculos adicionales
* Se registra la venta inmediatamente
* No afecta el efectivo en caja física

**Tarjeta de crédito:**

* Proceso directo similar a Bizum
* Se registra la venta inmediatamente
* No afecta el efectivo en caja física

**Efectivo:**

* **Proceso especial** con cálculo de cambio:
  1. Se presenta una pantalla adicional
  2. Campo para introducir: "¿Cuánto paga el cliente?"
  3. **Cálculo automático**: La aplicación calcula y muestra la devolución
  4. **Ejemplo**: Venta 17€, cliente paga 20€ → Devolución: 3€
  5. Confirmación antes de completar la venta

**4.3.4 Sistema de Anulación de Ventas**

**Principio fundamental:** El desarrollador estableció claramente que las ventas nunca se deben borrar de la base de datos. Esto es crucial para:

* Mantener integridad de datos
* Conservar trazabilidad completa
* Evitar inconsistencias en los arqueos
* Cumplir con buenas prácticas contables

**Proceso de anulación mediante "venta negativa":**

1. **Identificación**: El usuario mantiene pulsada una venta en la lista de ventas
2. **Confirmación**: El sistema pregunta: "¿Generar venta negativa?" (explicando que es una devolución)
3. **Creación automática**: Si se confirma, se crea una venta idéntica pero con todos los importes en negativo
4. **Método de pago**: La venta negativa usa automáticamente el mismo método de pago que la original
5. **Registro dual**: Ambas ventas (original y negativa) quedan permanentemente registradas

**Ejemplos prácticos:**

* **Venta original**: "Camiseta 20€ - Efectivo"
* **Venta negativa**: "Camiseta -20€ - Efectivo"
* **Resultado en arqueo**: Las dos ventas se cancelan mutuamente

**Visualización diferenciada:**

* **Ventas normales**: Se muestran en color estándar (negro o azul)
* **Ventas negativas**: Se muestran en color rojo para identificación inmediata
* **Posible iconografía**: Icono de devolución (↩️) para reforzar la identificación

**4.3.5 Movimientos de Efectivo en Caja (Funcionalidad Futura)**

Durante nuestra conversación se identificó la necesidad de registrar movimientos de efectivo que no son ventas ni gastos:

**Añadir dinero propio a la caja:**

* **Situación típica**: No hay suficiente cambio para los clientes
* **Ejemplo**: Solo tienes billetes de 20€ y necesitas dar cambio de una venta de 7€
* **Acción**: Añades billetes de 5€ y 10€ de tu dinero personal
* **Registro**: "Añadir efectivo para cambio: +50€"

**Retirar dinero de la caja:**

* **Situación típica**: Se ha acumulado demasiado dinero en efectivo (riesgo de seguridad)
* **Ejemplo**: Tienes 300€ en efectivo y decides guardar 200€ en un lugar más seguro
* **Acción**: Retiras dinero de la caja
* **Registro**: "Retirar efectivo por seguridad: -200€"

**Integración con arqueo:**

Efectivo final en caja = Saldo inicial + Ventas efectivo - Gastos efectivo + Dinero añadido - Dinero retirado

**4.4 Sistema de Arqueo**

**4.4.1 Concepto de Arqueo**

El desarrollador explicó que el arqueo es el proceso fundamental de cuadrar las cuentas al final del día de mercadillo. Consiste en comparar lo que debería haber en caja según las operaciones registradas con lo que realmente hay físicamente.

**4.4.2 Dos Tipos de Arqueo Diferenciados**

Se establecieron claramente dos cálculos diferentes que sirven para propósitos distintos:

**Arqueo de Caja Real (Dinero físico en efectivo):** Este cálculo determina exactamente cuánto dinero en efectivo debería haber físicamente en la caja:

Saldo Inicial en Efectivo

+ Ventas realizadas en Efectivo

- Gastos pagados en Efectivo

- Dinero Retirado durante el día (si aplica)

+ Dinero Añadido durante el día (si aplica)

= Efectivo que debe haber físicamente en la caja

**Saldo Total de Ventas del Día (Resultado económico completo):** Este cálculo muestra el resultado económico total del mercadillo, incluyendo todas las formas de pago:

Saldo Inicial

+ TODAS las Ventas (efectivo + tarjeta + bizum)

- TODOS los Gastos (efectivo + tarjeta + bizum)

= Resultado económico total del día

**4.4.3 Proceso de Arqueo Detallado**

El desarrollador describió cómo debería funcionar el proceso de arqueo:

**Paso 1: Finalización de la actividad comercial**

* El usuario indica en la aplicación que ha terminado de vender
* Se bloquean nuevas ventas para ese mercadillo

**Paso 2: Conteo físico del efectivo**

* El vendedor cuenta manualmente todo el dinero en efectivo de la caja
* Introduce el importe contado en la aplicación

**Paso 3: Comparación automática**

* La aplicación compara el dinero contado con el "arqueo de caja real" calculado
* Muestra claramente si hay diferencias (descuadres)

**Paso 4: Gestión de descuadres**

* Si hay diferencias, se registran como descuadre con motivo opcional
* Se permite continuar el proceso (los descuadres menores son normales)

**Paso 5: Cierre contable**

* Se registra oficialmente el arqueo en la base de datos
* El mercadillo queda marcado como "finalizado"
* Se procede a la gestión del saldo para el próximo mercadillo

**4.5 Gestión de Gastos**

**4.5.1 Filosofía de Simplicidad**

El desarrollador explicó que la gestión de gastos debe ser simple y directa, reconociendo que en un mercadillo los gastos suelen ser pocos, variados y no siguen patrones complejos.

**4.5.2 Proceso de Registro de Gastos**

**Características del sistema:**

* **Entrada individual**: Cada gasto se registra uno por uno cuando ocurre
* **Datos mínimos**: Solo descripción libre e importe
* **Formas de pago**: Efectivo, Bizum o Tarjeta (las mismas que las ventas)
* **Sin categorías predefinidas**: Máxima flexibilidad con descripción libre

**Ejemplos típicos de gastos:**

* "Gasolina ida y vuelta - 15€"
* "Desayuno bar mercadillo - 3.50€"
* "Parking zona comercial - 2€"
* "Material publicitario folletos - 8€"
* "Comida mediodía - 12€"

**4.5.3 Integración con el Arqueo**

Los gastos se integran automáticamente en ambos tipos de arqueo:

* **Gastos en efectivo**: Se restan del arqueo de caja real
* **Todos los gastos**: Se restan del saldo total de ventas del día

**4.6 Control de Stock (Versión Premium)**

**4.6.1 Arquitectura Flexible del Control de Stock**

Durante nuestra conversación se diseñó un sistema de control de stock con máxima flexibilidad, reconociendo que no todos los vendedores necesitan esta funcionalidad y que incluso quienes la usan pueden querer aplicarla solo a algunos productos.

**4.6.2 Configuración Global**

**Switch principal en configuración:**

* Ubicado en la tabla de configuración de la aplicación
* Permite activar o desactivar el control de stock para toda la aplicación
* **Ventajas**:
  + Usuarios que no necesitan control: Lo mantienen desactivado y la funcionalidad no molesta
  + Usuarios que sí lo necesitan: Lo activan y acceden a todas las funcionalidades

**4.6.3 Configuración por Artículo Individual**

Incluso con el control global activado, cada artículo puede tener configuración independiente:

**Checkbox "Controlar stock en este artículo":**

* Permite activar/desactivar el control para cada producto específicamente
* **Ejemplo de uso**: Controlar camisetas (cantidad limitada) pero no controlar servicios de personalización

**Campos adicionales cuando está activado:**

* **Unidades en stock**: Cantidad actual disponible del producto
* **Stock mínimo**: Nivel mínimo que genera avisos automáticos cuando se alcanza

**4.6.4 Funcionalidad de Venta sin Stock**

Se diseñó una funcionalidad especial para situaciones prácticas:

**Checkbox "Permitir venta sin stock" por artículo:**

* **Propósito**: Permitir ventas incluso cuando el stock sea cero o negativo
* **Caso de uso típico**: Situaciones de prisa en el mercadillo donde no hay tiempo de actualizar el inventario
* **Comportamiento**: Si está activado, se puede vender aunque stock = 0, quedando el inventario en negativo
* **Gestión posterior**: El vendedor ajusta el stock real cuando tiene tiempo

**4.6.5 Momento de Descuento de Stock**

Se estableció claramente cuándo se debe descontar el stock:

**Regla fundamental**: El stock se descuenta únicamente cuando una venta está completamente cerrada (forma de pago confirmada), no durante la construcción del ticket.

**Justificación**:

* Evita descuentos erróneos por ventas que se cancelan durante el proceso
* El cliente puede cambiar de opinión hasta el momento del pago
* Solo las ventas realmente completadas afectan el inventario

**Flujo detallado**:

1. Cliente selecciona productos → NO se descuenta stock
2. Se construye el ticket → NO se descuenta stock
3. Se confirma el ticket → NO se descuenta stock
4. Se selecciona forma de pago → NO se descuenta stock
5. Se completa el pago → SÍ se descuenta stock automáticamente

**4.7 Control de Costes (Versión Premium)**

**4.7.1 Funcionalidad de Análisis de Márgenes**

El control de costes fue diseñado para permitir análisis de rentabilidad por artículo, proporcionando información valiosa para la toma de decisiones comerciales.

**4.7.2 Configuración por Artículo**

**Sistema similar al control de stock:**

* **Checkbox "Controlar costes en este artículo"**: Permite activar la funcionalidad para cada producto específicamente
* **Campo precio de coste**: Solo aparece y es obligatorio si está activado el control
* **Cálculo automático**: La aplicación calcula automáticamente: Precio de venta - Precio de coste = Beneficio unitario

**4.7.3 Información que Proporcionará**

Una vez implementado completamente, el sistema proporcionará:

**A nivel de artículo individual:**

* Beneficio por unidad de cada producto
* Margen porcentual de cada artículo

**A nivel de venta:**

* Beneficio total por cada ticket de venta
* Desglose de márgenes por artículos dentro de cada venta

**A nivel de análisis:**

* Márgenes de beneficio por categoría de productos
* Identificación de artículos más rentables
* Identificación de artículos menos rentables
* Análisis de rentabilidad por períodos

**A nivel de mercadillo:**

* Beneficio bruto del día (suma de todos los beneficios de ventas)
* Beneficio neto del día (beneficio bruto - todos los gastos)
* Comparativa de rentabilidad entre mercadillos

**4.8 Sistema de Avisos**

**4.8.1 Arquitectura de Dos Niveles**

Durante nuestra conversación se diseñó un sistema de avisos sofisticado que utiliza una arquitectura de dos tablas para máxima flexibilidad y eficiencia:

**Tabla de Avisos Base (Catálogo de tipos):**

* Contiene los tipos de avisos estándar predefinidos por la aplicación
* No es configurable por el usuario final
* Define para cada tipo: título, mensaje base y prioridad
* Se inicializa automáticamente al crear la base de datos

**Tabla de Avisos Usuario (Instancias específicas):**

* Contiene las instancias específicas de avisos para el usuario actual
* Se crean y destruyen dinámicamente según el estado de la aplicación
* Pueden estar vinculados a mercadillos específicos o ser avisos generales
* Incluyen información contextual específica (fechas, importes, etc.)

**4.8.2 Tipos de Avisos por Comportamiento**

Se establecieron dos categorías fundamentales de avisos:

**Avisos Auto-descartables (Informativos):**

* Son avisos meramente informativos que notifican al usuario sobre situaciones
* El usuario puede marcarlos como leídos y desaparecen de la lista
* **Ejemplo**: "Tienes artículos con stock negativo"
* **Característica**: Se pueden resolver simplemente "tomando nota" de la información

**Avisos No Descartables (Acción requerida):**

* Son avisos que requieren una acción específica del usuario para resolverse
* NO se pueden marcar como leídos hasta que se resuelva el problema subyacente
* Persisten en la lista hasta que se ejecute la acción requerida
* **Ejemplos**:
  + "Tienes un arqueo de caja pendiente de cerrar"
  + "Hay un mercadillo anterior que sigue abierto"
* **Característica**: Solo desaparecen cuando se completa la acción correspondiente

**4.8.3 Interfaz de Usuario del Sistema de Avisos**

**Campanita de Notificaciones:** El desarrollador especificó detalladamente cómo debería funcionar:

* **Ubicación**: Icono de campana en la esquina superior derecha de todas las pantallas principales
* **Badge numérico**: Círculo rojo con el número de avisos no leídos
* **Comportamiento del badge**:
  + Se oculta completamente cuando no hay avisos pendientes (count = 0)
  + Muestra el número exacto hasta 9
  + Muestra "9+" cuando hay más de 9 avisos pendientes
* **Interacción**: Al tocar la campanita se abre el modal de avisos

**Modal de Lista de Avisos:**

* **Activación**: Se abre al tocar la campanita de notificaciones
* **Contenido**: Lista completa de todos los avisos pendientes, ordenados por fecha de creación (más recientes primero)
* **Información por aviso**: Título, mensaje descriptivo, fecha/hora de creación
* **Interacciones diferenciadas**:
  + Avisos descartables: Botón "Marcar como leído"
  + Avisos no descartables: Sin opción de descarte, solo visualización
* **Cierre**: Se puede cerrar tocando fuera del modal o botón de cerrar

**4.8.4 Ejemplos de Avisos Predefinidos Implementados**

Durante el análisis se identificaron y definieron avisos específicos:

**1. "Arqueo pendiente" (Prioridad 1, No descartable)**

* **Cuándo se genera**: Cuando hay un mercadillo que terminó su actividad comercial pero no se ha realizado el arqueo
* **Mensaje**: "No has cerrado el arqueo del último mercadillo"
* **Resolución**: Solo desaparece cuando se completa el proceso de arqueo

**2. "Mercadillo sin cerrar" (Prioridad 1, No descartable)**

* **Cuándo se genera**: Cuando hay un mercadillo que sigue marcado como "en curso" pasado un tiempo prudencial
* **Mensaje**: "Hay un mercadillo anterior que sigue abierto"
* **Resolución**: Solo desaparece cuando se cierra oficialmente el mercadillo

**3. "Stock bajo" (Prioridad 2, Descartable)**

* **Cuándo se genera**: Cuando artículos con control de stock alcanzan o superan el nivel mínimo configurado
* **Mensaje**: "Algunos artículos están bajo el nivel mínimo"
* **Resolución**: Se puede marcar como leído o desaparece cuando se actualiza el stock

**4.8.5 Sistema de Prioridades**

Se estableció un sistema de prioridades numéricas:

* **Prioridad 1**: Avisos críticos que requieren atención inmediata
* **Prioridad 2**: Avisos importantes pero no críticos
* **Prioridad 3**: Avisos informativos de baja importancia

**5. ESTRUCTURA DE DATOS Y ENTIDADES**

**5.1 Entidades Completamente Implementadas**

**5.1.1 CategoriaEntity**

Durante nuestra conversación se revisó la implementación completa de esta entidad:

**Campos y características:**

* **idCategoria (UUID)**: Identificador único para evitar conflictos en sincronización multiplataforma
* **nombre (String)**: Identificación de la categoría
* **colorHex (String)**: Color personalizable en formato hexadecimal para diferenciación visual
* **orden (Int)**: Permite reorganización manual por el usuario
* **activa (Boolean)**: Implementa "soft delete" para no perder datos históricos
* **pendienteSincronizar (Boolean)**: Flag para gestión de sincronización offline-first

**Funcionalidades implementadas:**

* CRUD completo con sincronización automática
* Sistema de colores personalizables con selector profesional
* Reordenamiento manual por el usuario
* Activación/desactivación sin pérdida de datos históricos
* Compatible con constructor vacío requerido por Firestore

**5.1.2 ConfiguracionEntity**

Se revisó la implementación de la configuración global:

**Concepto de registro único:**

* **id = 1**: Siempre fijo para garantizar una sola configuración global
* Simplifica la gestión al no tener múltiples registros de configuración

**Campos de versión y licenciamiento:**

* **versionApp**: FREE, BETA, PREMIUM (control de funcionalidades)
* **numeroVersion**: Versión funcional exacta de la aplicación
* **isConfigurated**: Indica si se completó el setup inicial

**Campos de funcionalidades Premium:**

* **controlStock (Int)**: 0 = desactivado, 1 = activado (más flexible que boolean)
* **controlCoste (Int)**: 0 = desactivado, 1 = activado

**Campos de autenticación y dispositivos:**

* **ultimoDispositivo**: ID del dispositivo autorizado (crucial para sistema monopuesto)
* **usuarioEmail**: Email del usuario autenticado
* **usuarioId**: UID en Firebase para funcionalidades Premium

**Campos de personalización:**

* **idioma**: Configuración de internacionalización
* **temaOscuro**: Preferencia de tema visual
* **fuente**: Tipografía seleccionada
* **moneda**: Formato de moneda para la aplicación

**Campos de sincronización:**

* **fechaUltimaSync**: Control de estado de sincronización con Firebase

**5.1.3 Sistema de Avisos (AvisoBaseEntity + AvisoUsuarioEntity)**

Se implementó un sistema completo de dos entidades relacionadas:

**AvisoBaseEntity (Catálogo de tipos):**

* **idAvisoBase (Int)**: ID numérico para tipos predefinidos
* **titulo**: Título estándar del tipo de aviso
* **mensaje**: Mensaje base que se mostrará al usuario
* **prioridad**: Nivel de importancia (1 = crítico, 2 = importante, 3 = informativo)

**AvisoUsuarioEntity (Instancias específicas):**

* **idAvisoUsuario (String UUID)**: Identificador único para sincronización
* **idAvisoBase (Int FK)**: Relación con el tipo de aviso base
* **fechaCreacion (Long)**: Timestamp de cuando se generó el aviso
* **leido (Boolean)**: Estado de lectura del aviso
* **pendienteSincronizar (Boolean)**: Para sincronización offline-first
* **mercadilloId (String opcional)**: Para avisos específicos de un mercadillo

**AvisoConDetalle (Relación completa):**

* Utiliza @Embedded y @Relation para obtener información completa
* Una sola consulta obtiene el aviso del usuario + detalles del tipo base
* Perfecto para mostrar en la interfaz del modal de avisos

**5.2 Entidades Documentadas Pendientes de Implementación**

Durante nuestra conversación se revisó la documentación técnica proporcionada, identificando las siguientes entidades que están completamente especificadas pero aún requieren implementación:

**5.2.1 MercadillosEntity**

**Campos principales identificados:**

* **id (String UUID)**: Identificador único
* **fechaHora (DateTime)**: Fecha y hora del mercadillo
* **lugar, organizador**: Información básica obligatoria
* **turno, ubicacion, municipio**: Información adicional opcional
* **esGratis, importeSuscripcion**: Datos económicos del puesto
* **puntoDeLuz, requiereCarpa, requiereMesa**: Datos logísticos
* **estado (Int)**: Estado del mercadillo (programado, en curso, finalizado, etc.)
* **saldoInicial (Double)**: Saldo de inicio del mercadillo

**5.2.2 ArticulosEntity**

**Campos principales identificados:**

* **idArticulo (String UUID)**: Identificador único
* **nombre**: Nombre del artículo
* **idCategoria (String FK)**: Relación con categorías
* **precioVenta (Double)**: Precio al público
* **activo (Boolean)**: Estado del artículo
* **stock (Int)**: Solo versión Premium si se activa control
* **controlarStock (Boolean)**: Activación individual del control
* **precioCoste (Double)**: Para control de márgenes (Premium)
* **controlarCoste (Boolean)**: Activación individual del control de costes
* **favorito (Boolean)**: Para accesos rápidos
* **fotoUri (String)**: Imagen del artículo (Premium)

**5.2.3 Sistema de Ventas Completo**

**VentasEntity:**

* **id (Integer PK)**: Clave primaria autoincremental
* **mercadilloId (Integer FK)**: Relación con mercadillo
* **fechaHora (Text)**: Timestamp de la venta
* **tipoPago**: Efectivo, tarjeta, bizum (sin mixto en v1.0)
* **importeTotal (Real)**: Total de la venta
* **observaciones**: Comentarios opcionales
* **usuarioId**: Para versión Premium multiusuario

**VentaDetalleEntity:**

* **id (Integer PK)**: Clave primaria
* **ventaId (Integer FK)**: Relación con venta principal
* **articuloId (Integer FK)**: Artículo vendido
* **cantidad (Integer)**: Unidades (puede ser negativa para devoluciones)
* **precioUnidad (Real)**: Precio en el momento de la venta
* **subtotal (Real)**: cantidad × precioUnidad
* **controlaStock (Boolean)**: Estado del control en momento de venta
* **esDevolucion (Boolean)**: Indica si es venta negativa

**PagosEntity:**

* **id (Integer PK)**: Clave primaria
* **ventaId (Integer FK)**: Relación con venta
* **tipoPago**: Efectivo, tarjeta, bizum
* **importe (Real)**: Importe de este método de pago

**5.2.4 Otras Entidades Documentadas**

**GastosEntity, ArticulosStockEntity, ArqueosEntity, UsuariosEntity**: Todas completamente especificadas en la documentación técnica con campos detallados, tipos de datos y relaciones.

**6. COMPONENTES DE INTERFAZ DE USUARIO**

**6.1 Componentes Básicos Implementados**

**6.1.1 BotonHome**

Se implementó un componente reutilizable para la pantalla principal:

**Características de diseño:**

* **Superficie con color personalizable**: Fondo configurable para diferenciación
* **Icono vectorial con tinte**: Iconos SVG que se adaptan al color del tema
* **Texto descriptivo**: Debajo del icono, con tipografía Material Design
* **Bordes redondeados**: 16dp para aspecto moderno
* **Dimensiones responsivas**: widthIn(140dp, 160dp) para adaptación
* **Área de toque adecuada**: 140dp height para accesibilidad

**Funcionalidad:**

* **Completamente configurable**: Icono, texto, colores y acción onclick
* **Material Design 3**: Integración completa con el sistema de colores del tema
* **Uso típico**: Los 4 botones principales de la pantalla Home (Mercadillos, Artículos y Categorías, Listados, Configuración)

**6.1.2 CampanaAvisos**

Componente especializado para el sistema de notificaciones:

**Funcionalidades implementadas:**

* **Icono de campana**: SVG personalizado tinteable
* **Badge dinámico**: Círculo rojo con número de avisos
* **Lógica inteligente del badge**:
  + Se oculta completamente cuando count = 0
  + Muestra número exacto del 1 al 9
  + Muestra "9+" cuando count > 9 (estándar de aplicaciones móviles)
* **Posicionamiento preciso**: Badge en esquina superior derecha con offset
* **Colores configurables**: Tanto el icono como el badge son personalizables
* **Área de toque**: 48dp para cumplir estándares de accesibilidad

**Integración con el sistema:**

* Se conecta directamente con el Flow de avisos no leídos
* Actualización en tiempo real cuando cambia el estado de los avisos
* Callback onClick para abrir el modal de avisos

**6.1.3 DialogSelectorColor**

Selector de color profesional implementado:

**Funcionalidades técnicas:**

* **Librería Skydoves ColorPicker**: Solución robusta y profesional
* **Rueda de colores HSV**: Selector principal intuitivo
* **Control de transparencia**: AlphaSlider para opacidad
* **Control de brillo**: BrightnessSlider para luminosidad
* **Preview en tiempo real**: El controlador actualiza automáticamente la selección

**Integración:**

* **Controller pattern**: rememberColorPickerController() maneja todo el estado
* **Callback simple**: Devuelve Color directamente sin conversiones complejas
* **Uso principal**: Personalización de colores de categorías

**6.1.4 CustomDialogConfirmacion**

Diálogo reutilizable para confirmaciones críticas:

**Características:**

* **AlertDialog estándar**: Cumple patrones de Material Design
* **Totalmente configurable**: Título, mensaje, texto del botón y colores
* **Dos botones**: Confirmar (configurable) y Cancelar (estándar)
* **Gestión de eventos**: onConfirmar y onCancelar bien separados
* **onDismissRequest**: Permite cerrar tocando fuera o con botón back

**Casos de uso identificados:**

* Eliminar categorías
* Cerrar mercadillos
* Confirmar arqueos
* Cualquier acción irreversible

**6.2 Filosofía de Diseño de Componentes**

**6.2.1 Material Design 3**

Todos los componentes implementados siguen estrictamente las directrices:

* **Sistema de colores**: Uso consistente de MaterialTheme.colorScheme
* **Tipografía**: MaterialTheme.typography para textos coherentes
* **Elevaciones y sombras**: tonalElevation apropiada
* **Formas**: RoundedCornerShape con radios consistentes

**6.2.2 Accesibilidad**

* **Áreas de toque mínimas**: 48dp para todos los elementos interactivos
* **Contraste adecuado**: Verificación entre colores de fondo y contenido
* **ContentDescription**: Descripciones para lectores de pantalla
* **Navegación por teclado**: Soporte cuando aplica

**6.2.3 Consistencia Visual**

* **Paleta de colores unificada**: Todos los componentes usan los mismos colores base
* **Espaciados consistentes**: Sistema de espaciado basado en múltiplos de 8dp
* **Formas uniformes**: Mismos radios de borde en toda la aplicación
* **Iconografía coherente**: Estilo consistente de iconos SVG

**7. PATRONES DE DESARROLLO Y ARQUITECTURA**

**7.1 Repository Pattern Implementado**

Cada entidad principal tiene su repositorio que implementa la misma estructura:

**Responsabilidades del Repository:**

* **Abstracción de fuentes de datos**: Oculta si los datos vienen de Room o Firebase
* **Implementación offline-first**: Toda operación va primero a Room
* **Sincronización automática**: Intenta subir a Firebase inmediatamente
* **Gestión de errores**: Maneja fallos de conectividad sin afectar la UI
* **Interfaces limpias**: Proporciona métodos simples a los ViewModels

**Patrón de operaciones típico:**

suspend fun insertItem(item: Entity) {

// 1. Guardar local primero (UI inmediata)

dao.insert(item.copy(pendienteSincronizar = true))

// 2. Intentar sincronización

try {

firestore.collection("items").document(item.id).set(item).await()

dao.marcarComoSincronizada(item.id)

} catch (\_: Exception) {

// Si falla, queda pendiente para próxima sincronización

}

}

**7.2 MVVM Architecture**

La arquitectura sigue estrictamente el patrón MVVM:

**Model (Capa de datos):**

* Entidades de Room que representan los datos
* Repositorios que gestionan el acceso a datos
* Lógica de negocio pura sin dependencias de UI

**View (Capa de presentación):**

* Composables de Jetpack Compose
* UI reactiva que se actualiza automáticamente
* Sin lógica de negocio, solo presentación

**ViewModel (Capa de lógica de presentación):**

* Gestión del estado de UI
* Exposición de datos mediante Flow
* Manejo de eventos de usuario
* Comunicación con repositorios

**7.3 Reactive Programming con Flow**

Se utiliza extensivamente programación reactiva:

**Flujos de datos reactivos:**

* Todos los datos se exponen como Flow desde los repositorios
* La UI se actualiza automáticamente cuando cambian los datos
* No hay necesidad de refrescar manualmente las pantallas

**Beneficios observados:**

* UI siempre sincronizada con los datos
* Menos errores por estados inconsistentes
* Código más limpio y mantenible

**7.4 Clean Architecture Principles**

La organización sigue principios de Clean Architecture:

**Separación de capas:**

* Capa de datos (Room + Firebase)
* Capa de dominio (Repositorios + Lógica de negocio)
* Capa de presentación (UI + ViewModels)

**Regla de dependencias:**

* Las dependencias apuntan hacia adentro
* La UI depende de ViewModels, no de datos directamente
* Los repositorios pueden cambiar sin afectar la UI

**8. ESTRATEGIA DE SINCRONIZACIÓN**

**8.1 Principios Fundamentales del Sistema**

Durante nuestra conversación se establecieron principios claros:

**Local First (Primero Local):**

* Toda operación se ejecuta primero en la base de datos local (Room)
* La UI recibe respuesta inmediata sin esperar operaciones de red
* Los datos están siempre disponibles, con o sin internet

**Sync When Possible (Sincronizar Cuando Sea Posible):**

* Si hay conectividad, se sincroniza inmediatamente con Firebase
* Si no hay conectividad, se marca como "pendiente de sincronizar"
* No se bloquea nunca la operación por problemas de red

**Never Block UI (Nunca Bloquear la UI):**

* La interfaz de usuario nunca espera por operaciones de red
* Todas las operaciones de red son asíncronas y en background
* El usuario puede seguir trabajando independientemente del estado de la red

**Graceful Degradation (Degradación Elegante):**

* La aplicación funciona completamente sin internet
* Las funcionalidades no se limitan por falta de conectividad
* La experiencia de usuario es prácticamente idéntica online y offline

**8.2 Mecánica Detallada de Sincronización**

**Fase 1: Operación Local**

1. El usuario ejecuta una acción (crear categoría, registrar venta, etc.)
2. Los datos se guardan inmediatamente en Room con pendienteSincronizar = true
3. La UI se actualiza instantáneamente mostrando los cambios
4. El usuario puede continuar trabajando inmediatamente

**Fase 2: Intento de Sincronización**

1. En paralelo, se verifica la conectividad a internet
2. Si hay conexión, se intenta subir los datos a Firebase Firestore
3. La operación de red se ejecuta en background sin afectar la UI

**Fase 3: Gestión del Resultado**

* **Si la sincronización es exitosa**: Se marca el registro como pendienteSincronizar = false
* **Si la sincronización falla**: El registro queda como pendienteSincronizar = true para reintento posterior

**Fase 4: Recuperación Automática**

1. La aplicación monitorea constantemente el estado de conectividad
2. Cuando se detecta que se ha recuperado la conexión a internet
3. Se ejecuta automáticamente un proceso de sincronización de todos los elementos pendientes
4. Este proceso es transparente e invisible para el usuario

**8.3 Gestión de Conflictos y Integridad**

**Uso de UUIDs:**

* Todas las entidades principales usan UUIDs como clave primaria
* Esto evita conflictos de ID entre dispositivos
* Permite generar IDs únicos offline sin riesgo de duplicados

**Timestamps para Resolución Temporal:**

* Cada registro incluye timestamps de creación y modificación
* Permite implementar estrategias de resolución basadas en "última escritura gana"

**Estrategia "Last Write Wins":**

* En caso de conflicto, prevalece la última modificación (por timestamp)
* Simplicidad de implementación y comprensión
* Apropiada para el contexto de uso (mayormente un solo usuario activo)

**8.4 Monitoreo y Recuperación**

**Detección de Conectividad:**

* Monitoreo continuo del estado de red del dispositivo
* Detección automática de recuperación de conexión
* No requiere intervención manual del usuario

**Proceso de Recuperación:**

1. Se identifica automáticamente cuando se recupera la conectividad
2. Se consultan todos los registros con pendienteSincronizar = true
3. Se intenta sincronizar cada registro pendiente
4. Se actualiza el estado de cada registro según el resultado
5. El proceso es completamente invisible para el usuario

**9. CONSIDERACIONES DE EXPERIENCIA DE USUARIO**

**9.1 Adaptación al Contexto de Uso**

Durante nuestra conversación se identificaron las particularidades del contexto de uso:

**Entorno de mercadillo:**

* Uso frecuente en exteriores con luz solar variable
* Posible uso con guantes en temporadas frías
* Necesidad de operación rápida durante picos de venta
* Posible uso con una sola mano mientras se atiende al cliente

**Adaptaciones implementadas:**

* **Botones de tamaño generoso**: Mínimo 48dp para facilitar el toque
* **Textos legibles**: Tamaños de fuente apropiados para lectura en exteriores
* **Colores con buen contraste**: Visibilidad en diferentes condiciones de luz
* **Flujos de trabajo optimizados**: Mínimo número de taps para operaciones comunes

**9.2 Flujo de Usuario Optimizado**

El flujo está diseñado para minimizar taps y maximizar eficiencia:

**Accesos directos:**

* Funciones más utilizadas accesibles desde la pantalla principal
* Navegación intuitiva y predecible
* Breadcrumbs claros para saber dónde se está

**Confirmaciones inteligentes:**

* Confirmaciones solo para acciones críticas e irreversibles
* Operaciones comunes sin confirmaciones innecesarias
* Opciones de deshacer cuando es posible

**Feedback visual inmediato:**

* Respuesta visual instantánea a toda interacción
* Estados de carga mínimos (gracias a la arquitectura offline-first)
* Indicadores claros del estado de sincronización cuando es relevante

**9.3 Internacionalización y Personalización**

Se ha preparado la base para personalización:

**Soporte multiidioma:**

* Configuración de idioma almacenada en ConfiguracionEntity
* Preparado para implementar strings.xml para diferentes idiomas
* Adaptación cultural de elementos UI según el mercado

**Personalización visual:**

* **Tema oscuro**: Configuración por usuario para diferentes preferencias
* **Fuente configurable**: Adaptación a diferentes necesidades de legibilidad
* **Colores de categorías**: Personalización completa del sistema de colores
* **Formato de moneda**: Adaptación a diferentes regiones y monedas

**10. CONSIDERACIONES DE SEGURIDAD Y PRIVACIDAD**

**10.1 Autenticación y Autorización**

El sistema de seguridad se basa en:

**OAuth con Google:**

* Delegación de autenticación a Google para máxima seguridad
* No almacenamiento de contraseñas en la aplicación
* Tokens de sesión gestionados automáticamente por Firebase

**Control de dispositivos:**

* Validación de dispositivo autorizado en cada operación crítica
* Revocación automática de acceso en dispositivos no autorizados
* Logs de acceso para monitoreo (en versión Premium)

**10.2 Protección de Datos**

**Datos locales:**

* Protección mediante permisos nativos de Android
* Base de datos local no accesible por otras aplicaciones
* No almacenamiento de información sensible en texto plano

**Comunicación con Firebase:**

* Todas las comunicaciones cifradas mediante HTTPS/TLS
* Reglas de seguridad de Firestore para acceso controlado
* Validación de permisos en el backend

**Backup y recuperación:**

* Sincronización automática como sistema principal de backup
* Exportación de datos para respaldo adicional (funcionalidad Premium)
* Recuperación completa desde la nube en caso de pérdida de dispositivo

**11. ROADMAP DETALLADO Y PRÓXIMOS PASOS**

**11.1 Fase 1 - Funcionalidades Core (Prioridad Alta)**

**1.1 Implementación de Autenticación Google**

* Setup completo de Firebase Authentication
* Configuración de OAuth con Google
* Implementación del flujo de login/logout
* Sistema de control de dispositivo monopuesto
* Detección y gestión de cambios de dispositivo
* Testing exhaustivo del sistema monopuesto

**1.2 Módulo de Artículos Completo**

* Implementación de ArticulosEntity según especificaciones
* CRUD completo de artículos
* Integración con sistema de categorías existente
* Interfaz de gestión de artículos
* Búsqueda y filtrado de artículos
* Sistema de favoritos para acceso rápido

**1.3 Gestión Básica de Mercadillos**

* Implementación de MercadillosEntity
* Alta y edición de mercadillos
* Vista de calendario con códigos de colores
* Estados básicos de mercadillo
* Gestión de saldos entre mercadillos
* Sistema de cancelación de mercadillos

**11.2 Fase 2 - Sistema de Ventas y Arqueo (Prioridad Alta)**

**2.1 Motor de Ventas Completo**

* Implementación de VentasEntity y VentaDetalleEntity
* Interfaz de venta con soporte para artículos predefinidos/manuales
* Sistema de tickets híbridos
* Cálculo automático de totales
* Gestión completa de formas de pago
* Sistema de ventas negativas para anulaciones

**2.2 Sistema de Arqueo**

* Implementación de ArqueosEntity
* Cálculos automáticos de arqueo real vs esperado
* Gestión de descuadres
* Integración con cierre de mercadillos
* Reportes de arqueo detallados

**2.3 Gestión de Gastos**

* Implementación de GastosEntity
* Interfaz simple para registro de gastos
* Integración con sistema de arqueo
* Categorización opcional de gastos

**11.3 Fase 3 - Funcionalidades Premium (Prioridad Media)**

**3.1 Control de Stock Avanzado**

* Implementación de ArticulosStockEntity
* Sistema de movimientos de inventario con trazabilidad
* Diferentes tipos de movimientos (venta, ajuste, devolución, entrada)
* Alertas automáticas de stock bajo
* Reportes de movimientos de stock
* Integración con sistema de ventas para descuento automático

**3.2 Control de Costes y Márgenes**

* Implementación completa del sistema de costes
* Cálculo automático de márgenes por artículo
* Análisis de rentabilidad por categoría
* Reportes de beneficios y márgenes
* Identificación de productos más/menos rentables
* Análisis comparativo entre mercadillos

**3.3 Sistema de Reportes Avanzado**

* Análisis de ventas por períodos (diario, semanal, mensual, anual)
* Comparativas entre mercadillos
* Evolución de beneficios en el tiempo
* Identificación de tendencias de venta
* Análisis de productos más vendidos
* Reportes de gastos por categorías

**11.4 Fase 4 - Pulimiento y Lanzamiento (Prioridad Media-Baja)**

**4.1 Optimización de Performance**

* Análisis y optimización de consultas de base de datos
* Implementación de paginación para listas grandes
* Optimización de sincronización (solo datos modificados)
* Caching inteligente de datos frecuentemente accedidos
* Reducción del tamaño de la APK

**4.2 Testing Exhaustivo**

* Testing unitario de repositorios y ViewModels
* Testing de integración del sistema de sincronización
* Testing de UI con Compose Testing
* Testing del sistema monopuesto con múltiples dispositivos
* Testing de rendimiento con grandes volúmenes de datos
* Testing de usabilidad con usuarios reales

**4.3 Documentación y Preparación para Lanzamiento**

* Documentación completa de usuario
* Guías de inicio rápido
* Videos tutoriales para funcionalidades principales
* Preparación de assets para Play Store
* Configuración de analytics y crash reporting
* Setup de sistema de feedback de usuarios

**11.5 Funcionalidades Futuras Identificadas (Backlog)**

**Mejoras de UI/UX:**

* Animaciones y transiciones mejoradas
* Modo nocturno optimizado para uso en exteriores
* Soporte para tablets con layout adaptativo
* Widgets para acceso rápido desde la pantalla principal

**Funcionalidades Avanzadas:**

* Lectura de códigos de barras con cámara
* Fotos de artículos con compresión automática
* Sistema de clientes frecuentes con historial
* Integración con sistemas de facturación
* Soporte para múltiples monedas
* Backup automático en Google Drive

**Integraciones:**

* Sincronización con plataformas de e-commerce
* Integración con sistemas de contabilidad
* APIs para terceros
* Exportación directa a gestores contables

**12. ESTADO ACTUAL DEL DESARROLLO**

**12.1 Funcionalidades Completamente Implementadas**

**Arquitectura Base Sólida:**

* Sistema de base de datos Room configurado y funcionando
* Integración con Firebase Firestore establecida
* Patrón Repository implementado con sincronización offline-first
* Arquitectura MVVM completamente funcional
* Sistema de navegación con Jetpack Compose Navigation

**Sistema de Categorías Completo:**

* CRUD completo de categorías funcionando
* Sincronización automática Room ↔ Firebase
* Selector de colores profesional integrado
* Reordenamiento manual de categorías
* Sistema de activación/desactivación (soft delete)
* Interfaz de usuario completa y pulida

**Sistema de Avisos Funcional:**

* Arquitectura de dos tablas completamente implementada
* Generación automática de avisos según el estado de la aplicación
* Campanita de notificaciones con badge dinámico
* Modal de avisos con diferenciación entre tipos
* Sistema de marcado como leído para avisos descartables
* Integración completa con el flujo de la aplicación

**Componentes UI Reutilizables:**

* BotonHome para pantalla principal
* CampanaAvisos para sistema de notificaciones
* DialogSelectorColor para personalización de colores
* CustomDialogConfirmacion para acciones críticas
* Todos siguiendo Material Design 3 y principios de accesibilidad

**12.2 Funcionalidades En Desarrollo**

**Sistema de Configuración:**

* **Estado actual**: Pantalla creada, tabla de configuración implementada con todos los campos necesarios
* **Pendiente**: Integración completa con Firebase Authentication
* **Pendiente**: Implementación del sistema monopuesto completo
* **Pendiente**: Flujo de configuración inicial (onboarding)

**Base de Datos:**

* **Estado actual**: 4 entidades completamente implementadas (Categorías, Configuración, Avisos Base, Avisos Usuario)
* **Pendiente**: Implementación de las 8 entidades restantes documentadas
* **Pendiente**: Relaciones entre entidades y foreign keys

**12.3 Funcionalidades Completamente Pendientes**

**Autenticación y Seguridad:**

* Setup completo de Firebase Authentication
* Implementación del flujo de login/logout con Google
* Sistema de control de dispositivos monopuesto
* Gestión de sesiones y tokens
* Validación de permisos por funcionalidad

**Módulo de Artículos:**

* Implementación de ArticulosEntity
* CRUD completo de artículos
* Relación con categorías existentes
* Interfaz de gestión de artículos
* Sistema de búsqueda y filtrado

**Gestión de Mercadillos:**

* Implementación de MercadillosEntity
* Vista de calendario con códigos de colores
* Sistema de estados de mercadillo
* Gestión de saldos entre mercadillos
* Proceso de cierre y cancelación

**Sistema de Ventas:**

* Implementación de entidades de ventas (Ventas, VentaDetalle, Pagos)
* Motor de ventas con artículos predefinidos/manuales
* Gestión de formas de pago con cálculo de cambio
* Sistema de ventas negativas para anulaciones
* Integración con control de stock

**Sistema de Arqueo:**

* Implementación de ArqueosEntity
* Cálculos automáticos de arqueo
* Gestión de descuadres
* Integración con cierre de mercadillos

**Funcionalidades Premium:**

* Control de stock avanzado
* Control de costes y márgenes
* Sistema de reportes
* Gestión multiusuario

**13. ANÁLISIS DE RIESGOS Y CONSIDERACIONES TÉCNICAS**

**13.1 Riesgos Técnicos Identificados**

**Complejidad de Sincronización:**

* **Riesgo**: La sincronización bidireccional Room ↔ Firebase puede generar conflictos complejos
* **Mitigación**: Sistema de timestamps y UUIDs ya implementado, estrategia "last write wins"
* **Estado**: Parcialmente mitigado, requiere testing exhaustivo

**Performance con Grandes Volúmenes:**

* **Riesgo**: Degradación de performance con muchos mercadillos/ventas
* **Mitigación**: Implementar paginación y caching inteligente
* **Estado**: Identificado, pendiente de implementación

**Gestión de Estado Offline/Online:**

* **Riesgo**: Inconsistencias entre estado local y remoto
* **Mitigación**: Arquitectura offline-first ya implementada
* **Estado**: Bien mitigado por la arquitectura actual

**13.2 Consideraciones de Escalabilidad**

**Crecimiento de Datos:**

* La aplicación debe manejar años de datos de mercadillos
* Necesidad de archivado/compresión de datos antiguos
* Optimización de consultas para historiales largos

**Múltiples Usuarios (Premium):**

* Sistema de permisos y roles
* Gestión de conflictos entre usuarios simultáneos
* Audit trail para acciones críticas

**13.3 Dependencias Externas**

**Firebase:**

* Dependencia crítica de Google Firebase
* Necesidad de plan de contingencia para cambios de pricing
* Monitoreo de quotas y límites de uso

**APIs de Google:**

* OAuth depende de servicios de Google
* Necesidad de mantener credenciales actualizadas
* Gestión de cambios en políticas de Google

**14. MODELO ECONÓMICO Y VIABILIDAD**

**14.1 Costos de Desarrollo Identificados**

**Costos de Infraestructura:**

* Firebase: Costos variables según uso (probablemente mínimos inicialmente)
* Google Play Store: $25 registro + 30% comisión en ventas
* Certificados y herramientas de desarrollo

**Costos de Mantenimiento:**

* Actualizaciones por cambios en Android
* Mantenimiento de compatibilidad con Firebase
* Soporte técnico a usuarios
* Nuevas funcionalidades y mejoras

**14.2 Estrategia de Monetización**

**Versión FREE como Acquisition:**

* Funcionalidad completa para atraer usuarios
* Sistema monopuesto que incentiva upgrade natural
* Word-of-mouth en comunidad de mercadillos

**Versión PREMIUM como Retention:**

* Funcionalidades realmente valiosas (stock, costes, reportes)
* Pago único atractivo vs suscripciones
* Valor claro y mesurable para el usuario

**14.3 Mercado Objetivo**

**Mercado Primario:**

* Vendedores ambulantes individuales en mercadillos artesanos
* Pequeños negocios familiares en ferias
* Artesanos que venden en múltiples ubicaciones

**Mercado Secundario:**

* Pequeños comercios que participan en ferias ocasionalmente
* Organizadores de mercadillos que necesitan herramientas para vendedores
* Cooperativas de artesanos

**15. CONCLUSIONES Y REFLEXIONES FINALES**

**15.1 Fortalezas del Proyecto**

**Experiencia Real como Base:**

* El proyecto surge de necesidades reales identificadas por experiencia directa
* Cada funcionalidad resuelve problemas específicos del sector
* Comprensión profunda del contexto de uso y limitaciones

**Arquitectura Técnica Sólida:**

* Sistema offline-first probado y robusto
* Tecnologías modernas y bien soportadas
* Patrones de desarrollo escalables y mantenibles

**Diferenciación Clara:**

* Especialización en mercadillos vs aplicaciones generalistas
* Modelo de negocio equilibrado entre accesibilidad y monetización
* Funcionalidades específicas no disponibles en competidores

**15.2 Desafíos Principales**

**Complejidad de Implementación:**

* Muchas funcionalidades interconectadas
* Sistema de sincronización complejo
* Necesidad de testing exhaustivo

**Mercado Nicho:**

* Mercado específico y potencialmente limitado
* Dependencia del crecimiento del sector artesano
* Necesidad de estrategia de marketing específica

**Competencia Potencial:**

* Riesgo de que grandes players introduzcan funcionalidades similares
* Necesidad de mantener ventaja competitiva con innovación continua

**15.3 Factores Críticos de Éxito**

**Calidad de Implementación:**

* Sistema que funcione de manera absolutamente confiable
* Performance excelente en condiciones reales de uso
* UI/UX adaptada específicamente al contexto de mercadillos

**Adopción del Mercado:**

* Penetración inicial en comunidades de artesanos
* Feedback temprano y mejora iterativa
* Construcción de reputación en el sector

**Evolución Continua:**

* Capacidad de adaptarse a cambios en el mercado
* Incorporación de feedback de usuarios reales
* Innovación continua en funcionalidades

**15.4 Visión a Largo Plazo**

El proyecto tiene potencial para evolucionar desde una aplicación de control de caja específica hacia una plataforma integral para vendedores ambulantes, incluyendo:

* Gestión completa de inventario multi-ubicación
* Herramientas de marketing y fidelización de clientes
* Integración con plataformas de venta online
* Análisis predictivo de ventas por ubicación/temporada
* Comunidad de vendedores con intercambio de información

La base técnica sólida y la comprensión profunda del mercado objetivo proporcionan una fundación excelente para esta evolución futura.

**ANEXO: ESTADO DE LA CONVERSACIÓN**

Este documento refleja el análisis completo y detallado desarrollado durante nuestra conversación sobre la aplicación "Caja Mercadillos". Durante el chat se establecieron:

* **Visión global** del proyecto antes de implementar funcionalidades específicas
* **Arquitectura técnica** basada en principios offline-first
* **Funcionalidades detalladas** pensadas desde la experiencia real de uso
* **Modelo de negocio** equilibrado entre accesibilidad y monetización
* **Roadmap estructurado** para desarrollo por fases
* **Consideraciones técnicas** para escalabilidad y mantenimiento

En lo hablado con la IA faltan algunas cosas, entre ellas la implementación de los recibos tras venta. Una vez finalizada la venta, si es Bizum o Tarjeta nada mas registrar el pago y si es efectivo tras indicar la vuelta, hay que generar un recibo y dar dos opciones, envio por correo o por sms/WhatsApp.  
Tambien se incluye para una versión futura 2.0, 3.0 etc, la posibilidad de integrar el propio pago con tarjeta en la app, usando SumUp o similares.

El diseño no esta completamente cerrado y seguramente se ira ampliando sobre la construcción de la aplicación. Siempre que se implemente una nueva funcionalidad se debe decidir si es para la versión gratuita o para la Premium. La IA deberá tener una lista de tareas pendientes actualizada.

Este documento se irá ampliando durante el desarrollo de la aplicación.

El desarrollador expresó específicamente la necesidad de tener esta visión completa para evitar el problema de "arreglar una cosa y estropear otra" que había experimentado con otras inteligencias artificiales. Este documento servirá como referencia completa para continuar el desarrollo manteniendo la coherencia y visión global del proyecto.

La conversación se cortó mientras se estaba revisando el código fuente actual de la aplicación, habiendo completado el análisis funcional y de arquitectura. El próximo paso sería continuar con la revisión del código implementado para identificar exactamente qué completar y en qué orden abordar las funcionalidades pendientes.

A partir de aquí viene el código fuente de la aplicación implementado hasta ahora. Puede que haya algún error en el código, de ser así lo pondré también aquí.  
  
package es.nuskysoftware.cajamercadillos.data.local.dao  
  
import androidx.room.\*  
import es.nuskysoftware.cajamercadillos.data.local.entity.AvisoBaseEntity  
import kotlinx.coroutines.flow.Flow  
  
@Dao  
interface AvisoBaseDao {  
 @Insert(onConflict = OnConflictStrategy.REPLACE)  
 suspend fun insertAvisosBase(lista: List<AvisoBaseEntity>)  
  
 @Query("SELECT \* FROM avisos\_base ORDER BY prioridad ASC")  
 fun getAvisosBase(): Flow<List<AvisoBaseEntity>>  
}

package es.nuskysoftware.cajamercadillos.data.local.dao  
  
import androidx.room.\*  
import es.nuskysoftware.cajamercadillos.data.local.entity.AvisoUsuarioEntity  
import es.nuskysoftware.cajamercadillos.data.local.entity.AvisoConDetalle  
import kotlinx.coroutines.flow.Flow  
  
@Dao  
interface AvisoUsuarioDao {  
 @Insert(onConflict = OnConflictStrategy.REPLACE)  
 suspend fun insertAvisoUsuario(aviso: AvisoUsuarioEntity)  
  
 @Transaction  
 @Query("SELECT \* FROM avisos\_usuario ORDER BY fechaCreacion DESC")  
 fun getAvisosConDetalle(): Flow<List<AvisoConDetalle>>  
  
 @Query("SELECT COUNT(\*) FROM avisos\_usuario WHERE leido = 0")  
 fun getAvisosNoLeidosCount(): Flow<Int>  
  
 @Query("UPDATE avisos\_usuario SET leido = 1 WHERE idAvisoUsuario = :idAviso")  
 suspend fun marcarAvisoComoLeido(idAviso: String)  
  
 @Query("DELETE FROM avisos\_usuario WHERE mercadilloId = :mercadilloId")  
 suspend fun deleteAvisosPorMercadillo(mercadilloId: String)  
  
 // ✅ AGREGADO: Mé*todo que faltaba para obtener avisos pendientes de sincronizar* @Query("SELECT \* FROM avisos\_usuario WHERE pendienteSincronizar = 1")  
 suspend fun getAvisosPendientes(): List<AvisoUsuarioEntity>  
  
 // ✅ AGREGADO: Mé*todo para marcar como sincronizado* @Query("UPDATE avisos\_usuario SET pendienteSincronizar = 0 WHERE idAvisoUsuario = :idAviso")  
 suspend fun marcarComoSincronizada(idAviso: String)  
}

package es.nuskysoftware.cajamercadillos.data.local.dao  
  
import androidx.room.\*  
import es.nuskysoftware.cajamercadillos.data.local.entity.CategoriaEntity  
import kotlinx.coroutines.flow.Flow  
  
*/\*\*  
 \* CategoriaDao:  
 \* Define todas las operaciones CRUD sobre la tabla "categorias".  
 \* Incluye métodos extra para sincronización con Firestore.  
 \*/*@Dao  
interface CategoriaDao {  
  
 // ------------------- INSERTAR -------------------  
 @Insert(onConflict = OnConflictStrategy.REPLACE)  
 suspend fun insertCategoria(categoria: CategoriaEntity)  
 */\*\*  
 \* Inserta una categoría en Room.  
 \* - Si ya existe (mismo idCategoria) → la reemplaza (estrategia REPLACE).  
 \* - Operación suspendida (corrutina).  
 \*/* // ------------------- ACTUALIZAR -------------------  
 @Update  
 suspend fun updateCategoria(categoria: CategoriaEntity)  
 */\*\*  
 \* Actualiza una categoría existente en la base local.  
 \* Necesita que idCategoria exista previamente.  
 \*/* // ------------------- ELIMINAR -------------------  
 @Delete  
 suspend fun deleteCategoria(categoria: CategoriaEntity)  
 */\*\*  
 \* Elimina una categoría específica.  
 \*/* // ------------------- OBTENER TODAS -------------------  
 @Query("SELECT \* FROM categorias ORDER BY orden ASC, nombre ASC")  
 fun getAllCategorias(): Flow<List<CategoriaEntity>>  
 */\*\*  
 \* Devuelve todas las categorías en tiempo real (Flow).  
 \* La UI se actualizará automáticamente cuando haya cambios en Room.  
 \* Ordenadas por orden manual primero, luego por nombre.  
 \*/* // ------------------- SINCRONIZACIÓN FIRESTORE -------------------  
  
 @Query("SELECT \* FROM categorias WHERE pendienteSincronizar = 1")  
 suspend fun getCategoriasPendientes(): List<CategoriaEntity>  
 */\*\*  
 \* Devuelve las categorías que aún NO están sincronizadas con Firestore.  
 \* Se usará en el repositorio para enviar cambios cuando haya conexión.  
 \*/* @Query("UPDATE categorias SET pendienteSincronizar = 0 WHERE idCategoria = :idCategoria")  
 suspend fun marcarComoSincronizada(idCategoria: String)  
 */\*\*  
 \* Marca como sincronizada una categoría (pendienteSincronizar = false)  
 \* cuando se confirma que fue subida correctamente a Firestore.  
 \*/*}

package es.nuskysoftware.cajamercadillos.data.local.dao  
  
import androidx.room.\*  
import es.nuskysoftware.cajamercadillos.data.local.entity.ConfiguracionEntity  
import kotlinx.coroutines.flow.Flow  
  
*/\*\*  
 \* ConfiguracionDao:  
 \* ✅ DAO para la tabla configuracion (registro único con id = 1).  
 \* ✅ Incluye métodos CRUD completos para la configuración global.  
 \*/*@Dao  
interface ConfiguracionDao {  
  
 // ------------------- INSERTAR/ACTUALIZAR -------------------  
 @Insert(onConflict = OnConflictStrategy.REPLACE)  
 suspend fun insertConfiguracion(config: ConfiguracionEntity)  
 */\*\*  
 \* Inserta o reemplaza la configuración global.  
 \* Como siempre usamos id = 1, REPLACE funciona como UPDATE.  
 \*/* // ------------------- OBTENER -------------------  
 @Query("SELECT \* FROM configuracion WHERE id = 1")  
 fun getConfiguracion(): Flow<ConfiguracionEntity?>  
 */\*\*  
 \* Devuelve la configuración global en tiempo real (Flow).  
 \* Si no existe, devuelve null.  
 \*/* @Query("SELECT \* FROM configuracion WHERE id = 1")  
 suspend fun getConfiguracionDirecta(): ConfiguracionEntity?  
 */\*\*  
 \* Versión suspendida (no Flow) para obtener configuración una sola vez.  
 \*/* // ------------------- ACTUALIZAR CAMPOS ESPECÍFICOS -------------------  
 @Query("UPDATE configuracion SET isConfigurated = :configurada WHERE id = 1")  
 suspend fun marcarComoConfigurada(configurada: Boolean)  
  
 @Query("UPDATE configuracion SET ultimoDispositivo = :deviceId WHERE id = 1")  
 suspend fun actualizarDispositivo(deviceId: String)  
  
 @Query("UPDATE configuracion SET usuarioEmail = :email, usuarioId = :uid WHERE id = 1")  
 suspend fun actualizarUsuario(email: String, uid: String)  
  
 @Query("UPDATE configuracion SET versionApp = :version WHERE id = 1")  
 suspend fun actualizarVersionApp(version: String)  
  
 @Query("UPDATE configuracion SET fechaUltimaSync = :fecha WHERE id = 1")  
 suspend fun actualizarFechaSync(fecha: String)  
  
 // ------------------- ELIMINAR (Raramente usado) -------------------  
 @Query("DELETE FROM configuracion WHERE id = 1")  
 suspend fun eliminarConfiguracion()  
 */\*\*  
 \* Elimina la configuración global.  
 \* Solo en casos extremos de reset completo.  
 \*/*}

package es.nuskysoftware.cajamercadillos.data.local.database  
  
import android.content.Context  
import androidx.room.Database  
import androidx.room.Room  
import androidx.room.RoomDatabase  
import androidx.sqlite.db.SupportSQLiteDatabase  
import es.nuskysoftware.cajamercadillos.data.local.dao.\*  
import es.nuskysoftware.cajamercadillos.data.local.entity.\*  
import kotlinx.coroutines.CoroutineScope  
import kotlinx.coroutines.Dispatchers  
import kotlinx.coroutines.launch  
  
@Database(  
 entities = [  
 CategoriaEntity::class,  
 ConfiguracionEntity::class,  
 AvisoBaseEntity::class,  
 AvisoUsuarioEntity::class  
 ],  
 version = 6,  
 exportSchema = true  
)  
abstract class AppDatabase : RoomDatabase() {  
  
 abstract fun categoriaDao(): CategoriaDao  
 abstract fun configuracionDao(): ConfiguracionDao  
 abstract fun avisoBaseDao(): AvisoBaseDao  
 abstract fun avisoUsuarioDao(): AvisoUsuarioDao  
  
 companion object {  
 @Volatile  
 private var INSTANCE: AppDatabase? = null  
  
 fun getDatabase(context: Context): AppDatabase {  
 return INSTANCE ?: *synchronized*(this) **{** val instance = Room.databaseBuilder(  
 context.*applicationContext*,  
 AppDatabase::class.*java*,  
 "caja\_mercadillos\_db"  
 )  
 .fallbackToDestructiveMigration(false)  
 .addCallback(object : Callback() {  
 override fun onCreate(db: SupportSQLiteDatabase) {  
 super.onCreate(db)  
 // ✅ INSERCIÓN DIRECTA con SQL para evitar problemas de recursión  
 db.execSQL("INSERT INTO configuracion (id, versionApp, numeroVersion, isConfigurated, controlStock, controlCoste, idioma, temaOscuro, fuente, moneda) VALUES (1, 'FREE', 'V1.0', 0, 0, 0, 'es', 0, 'Montserrat', '€ Euro')")  
  
 // ✅ INSERCIÓN DE AVISOS BASE con SQL  
 db.execSQL("INSERT INTO avisos\_base (idAvisoBase, titulo, mensaje, prioridad) VALUES (1, 'Arqueo pendiente', 'No has cerrado el arqueo del último mercadillo.', 1)")  
 db.execSQL("INSERT INTO avisos\_base (idAvisoBase, titulo, mensaje, prioridad) VALUES (2, 'Mercadillo sin cerrar', 'Hay un mercadillo anterior que sigue abierto.', 1)")  
 db.execSQL("INSERT INTO avisos\_base (idAvisoBase, titulo, mensaje, prioridad) VALUES (3, 'Stock bajo', 'Algunos artículos están bajo el nivel mínimo.', 2)")  
 }  
 })  
 .build()  
  
 // ✅ FORZAR INICIALIZACIÓN: Esto ejecuta onCreate si es necesario  
 *CoroutineScope*(Dispatchers.IO).*launch* **{** instance.configuracionDao().getConfiguracionDirecta()  
 **}** INSTANCE = instance  
 instance  
 **}** }  
 }  
}

package es.nuskysoftware.cajamercadillos.data.local.entity  
  
import androidx.room.Entity  
import androidx.room.PrimaryKey  
  
@Entity(tableName = "avisos\_base")  
data class AvisoBaseEntity(  
 @PrimaryKey val idAvisoBase: Int,  
 val titulo: String,  
 val mensaje: String,  
 val prioridad: Int  
)

package es.nuskysoftware.cajamercadillos.data.local.entity  
  
import androidx.room.Embedded  
import androidx.room.Relation  
  
data class AvisoConDetalle(  
 @Embedded val avisoUsuario: AvisoUsuarioEntity,  
 @Relation(  
 parentColumn = "idAvisoBase",  
 entityColumn = "idAvisoBase"  
 )  
 val avisoBase: AvisoBaseEntity  
)

package es.nuskysoftware.cajamercadillos.data.local.entity  
  
import androidx.room.Entity  
import androidx.room.ForeignKey  
import androidx.room.PrimaryKey  
  
@Entity(  
 tableName = "avisos\_usuario",  
 foreignKeys = [  
 ForeignKey(  
 entity = AvisoBaseEntity::class,  
 parentColumns = ["idAvisoBase"],  
 childColumns = ["idAvisoBase"],  
 onDelete = ForeignKey.CASCADE  
 )  
 ]  
)  
data class AvisoUsuarioEntity(  
 @PrimaryKey val idAvisoUsuario: String,  
 val idAvisoBase: Int,  
 val fechaCreacion: Long,  
 val leido: Boolean = false,  
 val pendienteSincronizar: Boolean = true,  
 val mercadilloId: String? = null  
)

package es.nuskysoftware.cajamercadillos.data.local.entity  
  
import androidx.room.Entity  
import androidx.room.PrimaryKey  
import java.util.UUID  
  
*/\*\*  
 \* CategoriaEntity:  
 \* ✅ Representa una categoría en la base de datos local (Room) y en Firestore.  
 \* ✅ Compatible con sincronización híbrida:  
 \* - Room: persistencia offline.  
 \* - Firestore: sincronización en la nube.  
 \* ✅ Incluye constructor vacío obligatorio para Firestore.  
 \*/*@Entity(tableName = "categorias")  
data class CategoriaEntity(  
  
 @PrimaryKey  
 val idCategoria: String = UUID.randomUUID().toString(),  
 // ✅ ID único (UUID) para evitar conflictos entre Room y Firestore.  
  
 val nombre: String = "",  
 // ✅ Nombre de la categoría. Ejemplo: "Frutas", "Bebidas".  
  
 val colorHex: String = "#FFFFFF",  
 // ✅ Color en formato HEX. Ejemplo: "#FF5733".  
  
 val orden: Int = 0,  
 // ✅ Orden manual de visualización (por defecto = 0).  
  
 val activa: Boolean = true,  
 // ✅ Estado lógico: true = activa, false = deshabilitada.  
  
 val pendienteSincronizar: Boolean = true  
 // ✅ Flag para saber si debe sincronizarse con Firestore.  
) {  
 // 🔑 Firestore necesita un constructor vacío explícito.  
 constructor() : this(  
 idCategoria = "",  
 nombre = "",  
 colorHex = "#FFFFFF",  
 orden = 0,  
 activa = true,  
 pendienteSincronizar = true  
 )  
}

package es.nuskysoftware.cajamercadillos.data.local.entity  
  
import androidx.room.Entity  
import androidx.room.PrimaryKey  
  
*/\*\*  
 \* ConfiguracionEntity:  
 \* ✅ Define la configuración global de la app (solo 1 registro en la tabla).  
 \*/*@Entity(tableName = "configuracion")  
data class ConfiguracionEntity(  
 @PrimaryKey  
 val id: Int = 1, // Siempre será 1 → una sola fila en la tabla.  
  
 val versionApp: String = "FREE", // FREE | BETA | PREMIUM  
 val numeroVersion: String = "V1.0", // Versión funcional exacta  
 val isConfigurated: Boolean = false, // ¿Configuración inicial completada?  
  
 val controlStock: Int = 0, // 0 = NO, 1 = SÍ (Premium)  
 val controlCoste: Int = 0, // 0 = NO, 1 = SÍ (Premium)  
  
 val ultimoDispositivo: String? = null, // ID único del dispositivo permitido  
 val usuarioEmail: String? = null, // Email del usuario (Premium)  
 val usuarioId: String? = null, // UID en Firebase (Premium)  
  
 val idioma: String = "es", // Idioma por defecto (es/en)  
 val temaOscuro: Boolean = false, // Modo oscuro activado o no  
 val fuente: String = "Montserrat", // Fuente por defecto  
 val moneda: String = "€ Euro", // Moneda por defecto  
  
 val fechaUltimaSync: String? = null // Fecha última sincronización (ISO 8601)  
)

package es.nuskysoftware.cajamercadillos.data.repository  
  
import com.google.firebase.firestore.FirebaseFirestore  
import com.google.firebase.firestore.SetOptions  
import es.nuskysoftware.cajamercadillos.data.local.dao.AvisoBaseDao  
import es.nuskysoftware.cajamercadillos.data.local.dao.AvisoUsuarioDao  
import es.nuskysoftware.cajamercadillos.data.local.entity.AvisoBaseEntity  
import es.nuskysoftware.cajamercadillos.data.local.entity.AvisoUsuarioEntity  
import es.nuskysoftware.cajamercadillos.data.local.entity.AvisoConDetalle  
import kotlinx.coroutines.flow.Flow  
import kotlinx.coroutines.tasks.await  
  
class AvisoRepository(  
 private val avisoBaseDao: AvisoBaseDao,  
 private val avisoUsuarioDao: AvisoUsuarioDao,  
 private val firestore: FirebaseFirestore  
) {  
  
 // ------------------- Avisos Base -------------------  
  
 suspend fun insertAvisosBase(lista: List<AvisoBaseEntity>) {  
 avisoBaseDao.insertAvisosBase(lista)  
 }  
  
 fun getAvisosBase(): Flow<List<AvisoBaseEntity>> {  
 return avisoBaseDao.getAvisosBase()  
 }  
  
 // ------------------- Avisos Usuario -------------------  
  
 suspend fun insertAvisoUsuario(aviso: AvisoUsuarioEntity) {  
 avisoUsuarioDao.insertAvisoUsuario(aviso)  
 subirAvisoAFirestore(aviso) // Subir inmediatamente  
 }  
  
 fun getAvisosConDetalle(): Flow<List<AvisoConDetalle>> {  
 return avisoUsuarioDao.getAvisosConDetalle()  
 }  
  
 suspend fun marcarAvisoComoLeido(idAviso: String) {  
 avisoUsuarioDao.marcarAvisoComoLeido(idAviso)  
 actualizarAvisoLeidoEnFirestore(idAviso)  
 }  
  
 fun getAvisosNoLeidosCount(): Flow<Int> {  
 return avisoUsuarioDao.getAvisosNoLeidosCount()  
 }  
  
 suspend fun eliminarAvisosPorMercadillo(mercadilloId: String) {  
 avisoUsuarioDao.deleteAvisosPorMercadillo(mercadilloId)  
 eliminarAvisosPorMercadilloEnFirestore(mercadilloId)  
 }  
  
 // ------------------- Sincronización Firestore -------------------  
  
 private fun subirAvisoAFirestore(aviso: AvisoUsuarioEntity) {  
 firestore.collection("avisos\_usuario")  
 .document(aviso.idAvisoUsuario) // ✅ CORREGIDO: usar idAvisoUsuario  
 .set(aviso, SetOptions.merge())  
 .addOnSuccessListener **{** *println*(">>> [Firestore] Aviso subido correctamente: ${aviso.idAvisoUsuario}")  
 **}** .addOnFailureListener **{** *println*(">>> [Firestore] Error al subir aviso: ${**it**.message}")  
 **}** }  
  
 private fun actualizarAvisoLeidoEnFirestore(idAviso: String) {  
 firestore.collection("avisos\_usuario")  
 .document(idAviso)  
 .update("leido", true)  
 .addOnSuccessListener **{** *println*(">>> [Firestore] Aviso marcado como leído en Firestore: $idAviso")  
 **}** .addOnFailureListener **{** *println*(">>> [Firestore] Error al actualizar aviso: ${**it**.message}")  
 **}** }  
  
 private fun eliminarAvisosPorMercadilloEnFirestore(mercadilloId: String) {  
 firestore.collection("avisos\_usuario")  
 .whereEqualTo("mercadilloId", mercadilloId)  
 .get()  
 .addOnSuccessListener **{** query **->** for (doc in query.*documents*) {  
 doc.*reference*.delete()  
 }  
 *println*(">>> [Firestore] Avisos eliminados para mercadillo: $mercadilloId")  
 **}** .addOnFailureListener **{** *println*(">>> [Firestore] Error al eliminar avisos: ${**it**.message}")  
 **}** }  
  
 */\*\*  
 \* Descarga todos los avisos desde Firestore y los guarda en Room.  
 \*/* suspend fun sincronizarDesdeFirestore() {  
 try {  
 val snapshot = firestore.collection("avisos\_usuario").get().await()  
 val listaAvisos = snapshot.*documents*.*mapNotNull* **{ it**.toObject(AvisoUsuarioEntity::class.*java*) **}** listaAvisos.*forEach* **{** avisoUsuarioDao.insertAvisoUsuario(**it**) **}** *println*(">>> [Firestore] Descargados ${listaAvisos.size} avisos desde Firestore.")  
 } catch (e: Exception) {  
 *println*(">>> [Firestore] Error al descargar avisos: ${e.message}")  
 }  
 }  
  
 */\*\*  
 \* Sube todos los avisos locales que aún no estén sincronizados.  
 \*/* suspend fun sincronizarPendientes() {  
 try {  
 // ✅ CORREGIDO: Obtener avisos pendientes desde Room  
 val listaLocal = avisoUsuarioDao.getAvisosPendientes() // Necesitamos implementar este mé*todo* listaLocal.*forEach* **{** subirAvisoAFirestore(**it**) **}** *println*(">>> [Firestore] Sincronización completada: ${listaLocal.size} avisos.")  
 } catch (e: Exception) {  
 *println*(">>> [Firestore] Error en sincronización: ${e.message}")  
 }  
 }  
}

package es.nuskysoftware.cajamercadillos.data.repository  
  
import com.google.firebase.firestore.FirebaseFirestore  
import es.nuskysoftware.cajamercadillos.data.local.dao.CategoriaDao  
import es.nuskysoftware.cajamercadillos.data.local.entity.CategoriaEntity  
import kotlinx.coroutines.flow.Flow  
import kotlinx.coroutines.tasks.await  
  
*/\*\*  
 \* ✅ CategoriaRepository:  
 \* - Gestiona sincronización híbrida (Room + Firestore).  
 \* - Inserta y actualiza primero en Room → UI inmediata.  
 \* - Sincroniza con Firestore en segundo plano.  
 \* - Si falla la conexión, marca los datos como pendientes.  
 \*/*class CategoriaRepository(  
 private val dao: CategoriaDao,  
 private val firestore: FirebaseFirestore  
) {  
  
 */\*\*  
 \* ✅ Devuelve todas las categorías en tiempo real desde Room.  
 \*/* fun getAllCategorias(): Flow<List<CategoriaEntity>> = dao.getAllCategorias()  
  
 */\*\*  
 \* ✅ Insertar nueva categoría:  
 \* 1️⃣ Guarda en Room (pendiente sincronizar).  
 \* 2️⃣ Intenta subir a Firestore.  
 \*/* suspend fun insertCategoria(categoria: CategoriaEntity) {  
 dao.insertCategoria(categoria.copy(pendienteSincronizar = true))  
 try {  
 firestore.collection("categorias")  
 .document(categoria.idCategoria)  
 .set(categoria)  
 .await()  
 dao.marcarComoSincronizada(categoria.idCategoria)  
 } catch (\_: Exception) {  
 // ❌ Si falla, se mantiene como pendiente  
 }  
 }  
  
 */\*\*  
 \* ✅ Actualizar categoría existente.  
 \*/* suspend fun updateCategoria(categoria: CategoriaEntity) {  
 dao.updateCategoria(categoria.copy(pendienteSincronizar = true))  
 try {  
 firestore.collection("categorias")  
 .document(categoria.idCategoria)  
 .set(categoria)  
 .await()  
 dao.marcarComoSincronizada(categoria.idCategoria)  
 } catch (\_: Exception) {  
 }  
 }  
  
 */\*\*  
 \* ✅ Eliminar categoría (Room + Firestore).  
 \*/* suspend fun deleteCategoria(categoria: CategoriaEntity) {  
 dao.deleteCategoria(categoria)  
 try {  
 firestore.collection("categorias")  
 .document(categoria.idCategoria)  
 .delete()  
 .await()  
 } catch (\_: Exception) {  
 // Futuro: marcar para eliminación pendiente  
 }  
 }  
  
 */\*\*  
 \* ✅ Sincronizar datos pendientes (Room → Firestore).  
 \*/* suspend fun sincronizarPendientes() {  
 val pendientes = dao.getCategoriasPendientes()  
 for (categoria in pendientes) {  
 try {  
 firestore.collection("categorias")  
 .document(categoria.idCategoria)  
 .set(categoria)  
 .await()  
 dao.marcarComoSincronizada(categoria.idCategoria)  
 } catch (\_: Exception) {  
 // Si falla, lo dejamos pendiente  
 }  
 }  
 }  
  
 */\*\*  
 \* ✅ Sincronizar Firestore → Room:  
 \* Descarga todas las categorías y las guarda localmente.  
 \*/* suspend fun sincronizarDesdeFirestore() {  
 try {  
 val snapshot = firestore.collection("categorias").get().await()  
 *println*(">>> [DEBUG] Firestore docs count: ${snapshot.size()}")  
 for (doc in snapshot.*documents*) {  
 *println*(">>> [DEBUG] Firestore doc: ${doc.*data*}")  
 val categoria = doc.toObject(CategoriaEntity::class.*java*)  
 if (categoria != null) {  
 *println*(">>> [DEBUG] Categoria parseada: $categoria")  
 dao.insertCategoria(categoria.copy(pendienteSincronizar = false))  
 } else {  
 *println*(">>> [ERROR] No se pudo parsear la categoría")  
 }  
 }  
 } catch (e: Exception) {  
 *println*(">>> [ERROR] sincronizarDesdeFirestore: ${e.message}")  
 }  
 }  
}

package es.nuskysoftware.cajamercadillos.data.repository  
  
import com.google.firebase.firestore.FirebaseFirestore  
import es.nuskysoftware.cajamercadillos.data.local.dao.ConfiguracionDao  
import es.nuskysoftware.cajamercadillos.data.local.entity.ConfiguracionEntity  
import kotlinx.coroutines.flow.Flow  
import kotlinx.coroutines.tasks.await  
  
class ConfiguracionRepository(  
 private val configuracionDao: ConfiguracionDao,  
 private val firestore: FirebaseFirestore  
) {  
 fun getConfiguracion(): Flow<ConfiguracionEntity?> = configuracionDao.getConfiguracion()  
  
 suspend fun guardarConfiguracion(config: ConfiguracionEntity) {  
 try {  
 configuracionDao.insertConfiguracion(config) // REPLACE asegura actualización  
 firestore.collection("configuracion")  
 .document("global")  
 .set(config)  
 .await()  
 } catch (\_: Exception) {  
 }  
 }  
  
 suspend fun sincronizarDesdeFirestore() {  
 try {  
 val snapshot = firestore.collection("configuracion")  
 .document("global")  
 .get()  
 .await()  
  
 if (snapshot.exists()) {  
 val config = snapshot.toObject(ConfiguracionEntity::class.*java*)  
 if (config != null) {  
 configuracionDao.insertConfiguracion(config)  
 }  
 }  
 } catch (\_: Exception) {  
 }  
 }  
}

package es.nuskysoftware.cajamercadillos.ui.components  
  
import androidx.compose.foundation.Image  
import androidx.compose.foundation.clickable  
import androidx.compose.foundation.layout.\*  
import androidx.compose.foundation.shape.RoundedCornerShape  
import androidx.compose.material3.MaterialTheme  
import androidx.compose.material3.Surface  
import androidx.compose.material3.Text  
import androidx.compose.runtime.Composable  
import androidx.compose.ui.Alignment  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.graphics.Color  
import androidx.compose.ui.graphics.ColorFilter  
import androidx.compose.ui.res.painterResource  
import androidx.compose.ui.text.style.TextAlign  
import androidx.compose.ui.unit.dp  
import androidx.compose.ui.unit.sp  
  
*/\*\*  
 \* BotonHome:  
 \* Composable reutilizable para mostrar un botón con:  
 \* - Un fondo coloreado  
 \* - Un icono centrado  
 \* - Texto debajo  
 \*  
 \* Parámetros:  
 \* @param iconRes: recurso del icono (SVG vector en drawable)  
 \* @param texto: texto que se muestra debajo del icono  
 \* @param colorFondo: color del fondo del botón  
 \* @param colorTexto: color del texto  
 \* @param onClick: acción al pulsar el botón  
 \*/*@Composable  
fun BotonHome(  
 iconRes: Int,  
 texto: String,  
 colorFondo: Color,  
 colorTexto: Color = MaterialTheme.colorScheme.onBackground,  
 onClick: () -> Unit  
) {  
 Column(  
 modifier = Modifier  
 .*widthIn*(min = 140.*dp*, max = 160.*dp*)  
 .*height*(140.*dp*)  
 .*clickable*(onClick = onClick),  
 horizontalAlignment = Alignment.CenterHorizontally,  
 verticalArrangement = Arrangement.Center  
 ) **{** // Contenedor del icono con un fondo coloreado  
 Surface(  
 modifier = Modifier.*size*(100.*dp*),  
 color = colorFondo,  
 shape = *RoundedCornerShape*(16.*dp*)  
 ) **{** // Icono en blanco sobre el fondo  
 Image(  
 painter = painterResource(id = iconRes),  
 contentDescription = texto,  
 modifier = Modifier  
 .*padding*(20.*dp*)  
 .*size*(60.*dp*),  
 colorFilter = ColorFilter.tint(MaterialTheme.colorScheme.onPrimary)  
 )  
 **}** Spacer(modifier = Modifier.*height*(8.*dp*))  
 // Texto centrado debajo del icono  
 Text(  
 text = texto,  
 style = MaterialTheme.typography.bodyMedium,  
 color = colorTexto,  
 textAlign = TextAlign.Center,  
 lineHeight = 16.*sp*,  
 modifier = Modifier.*fillMaxWidth*()  
 )  
 **}**}

package es.nuskysoftware.cajamercadillos.ui.components  
  
import androidx.compose.foundation.background  
import androidx.compose.foundation.clickable  
import androidx.compose.foundation.layout.\*  
import androidx.compose.foundation.shape.*CircleShape*import androidx.compose.material3.\*  
import androidx.compose.runtime.Composable  
import androidx.compose.ui.Alignment  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.graphics.Color  
import androidx.compose.ui.res.painterResource  
import androidx.compose.ui.unit.dp  
import es.nuskysoftware.cajamercadillos.R  
  
*/\*\*  
 \* CampanaAvisos:  
 \* ✅ Icono de campana con badge dinámico (número de avisos no leídos).  
 \* - colorIcono: color de la campana.  
 \* - colorFondoBadge: color del badge.  
 \* - count: número de avisos no leídos (si es 0, no se muestra el badge).  
 \* - onClick: acción al pulsar la campana.  
 \*/*@Composable  
fun CampanaAvisos(  
 count: Int,  
 colorIcono: Color = MaterialTheme.colorScheme.onBackground,  
 colorFondoBadge: Color = MaterialTheme.colorScheme.error,  
 onClick: () -> Unit  
) {  
 Box(  
 modifier = Modifier  
 .*size*(48.*dp*)  
 .*clickable* **{** onClick() **}**,  
 contentAlignment = Alignment.Center  
 ) **{** Icon(  
 painter = painterResource(id = R.drawable.*campana\_notificacion*), // Tu icono SVG personalizado  
 contentDescription = "Notificaciones",  
 tint = colorIcono,  
 modifier = Modifier.*size*(28.*dp*)  
 )  
  
 if (count > 0) {  
 Box(  
 modifier = Modifier  
 .*align*(Alignment.TopEnd)  
 .*offset*(x = 4.*dp*, y = (-4).*dp*)  
 .*size*(18.*dp*)  
 .*background*(colorFondoBadge, *CircleShape*),  
 contentAlignment = Alignment.Center  
 ) **{** Text(  
 text = if (count > 9) "9+" else count.toString(),  
 color = Color.White,  
 style = MaterialTheme.typography.labelSmall  
 )  
 **}** }  
 **}**}

package es.nuskysoftware.cajamercadillos.ui.components  
  
  
import androidx.compose.foundation.layout.\*  
import androidx.compose.foundation.shape.RoundedCornerShape  
import androidx.compose.material3.\*  
import androidx.compose.runtime.Composable  
import androidx.compose.ui.Alignment  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.graphics.Color  
import androidx.compose.ui.text.style.TextAlign  
import androidx.compose.ui.unit.dp  
  
*/\*\*  
 \* CustomDialogConfirmacion:  
 \* Diálogo reutilizable para confirmar acciones (ej. eliminar un ítem).  
 \*  
 \* Parámetros:  
 \* @param titulo -> Texto en la parte superior del diálogo.  
 \* @param mensaje -> Mensaje explicativo (ej. "¿Seguro que deseas eliminar...?").  
 \* @param textoConfirmar -> Texto del botón principal (ej. "Eliminar").  
 \* @param colorConfirmar -> Color del botón principal (ej. rojo para eliminar).  
 \* @param onConfirmar -> Acción cuando el usuario confirma.  
 \* @param onCancelar -> Acción cuando el usuario cancela o cierra el diálogo.  
 \*/*@Composable  
fun CustomDialogConfirmacion(  
 titulo: String,  
 mensaje: String,  
 textoConfirmar: String,  
 colorConfirmar: androidx.compose.ui.graphics.Color,  
 onConfirmar: () -> Unit,  
 onCancelar: () -> Unit  
) {  
 AlertDialog(  
 onDismissRequest = **{** onCancelar() **}**, // Si toca fuera o pulsa back → cancelar  
 confirmButton = **{** Button(  
 onClick = onConfirmar,  
 colors = ButtonDefaults.buttonColors(containerColor = colorConfirmar)  
 ) **{** Text(textoConfirmar, color =Color.Black)  
 **}  
 }**,  
 dismissButton = **{** OutlinedButton(onClick = onCancelar) **{** Text("Cancelar", color = Color.Black)  
 **}  
 }**,  
 title = **{** Text(  
 text = titulo,  
 style = MaterialTheme.typography.titleLarge,  
 textAlign = TextAlign.Center  
 )  
 **}**,  
 text = **{** Text(  
 text = mensaje,  
 style = MaterialTheme.typography.bodyMedium,  
 textAlign = TextAlign.Center  
 )  
 **}**,  
 shape = *RoundedCornerShape*(16.*dp*),  
 tonalElevation = 4.*dp*,  
 modifier = Modifier.*padding*(8.*dp*)  
 )  
}

package es.nuskysoftware.cajamercadillos.ui.components  
  
import androidx.compose.foundation.layout.\*  
import androidx.compose.material3.\*  
import androidx.compose.runtime.Composable  
import androidx.compose.ui.Alignment  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.graphics.Color  
import androidx.compose.ui.unit.dp  
import com.github.skydoves.colorpicker.compose.AlphaSlider  
import com.github.skydoves.colorpicker.compose.BrightnessSlider  
import com.github.skydoves.colorpicker.compose.HsvColorPicker  
import com.github.skydoves.colorpicker.compose.rememberColorPickerController  
  
*/\*\*  
 \* DialogSelectorColor:  
 \* Diálogo reutilizable con selector de color basado en Skydoves.  
 \* Incluye:  
 \* - HsvColorPicker (selector principal)  
 \* - Sliders de brillo y opacidad  
 \* - Botones Aceptar / Cancelar  
 \*/*@Composable  
fun DialogSelectorColor(  
 onColorElegido: (Color) -> Unit,  
 onCancelar: () -> Unit  
) {  
 val controller = rememberColorPickerController()  
  
 AlertDialog(  
 onDismissRequest = **{** onCancelar() **}**,  
 confirmButton = **{** TextButton(onClick = **{** onColorElegido(controller.selectedColor.value)  
 **}**) **{** Text("Aceptar", color = Color.Black)  
 **}  
 }**,  
 dismissButton = **{** TextButton(onClick = onCancelar) **{** Text("Cancelar", color = Color.Black)  
 **}  
 }**,  
 title = **{** Text("Selecciona un color") **}**,  
 text = **{** Column(horizontalAlignment = Alignment.CenterHorizontally) **{** // Selector principal  
 HsvColorPicker(  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*height*(200.*dp*)  
 .*padding*(8.*dp*),  
 controller = controller,  
 onColorChanged = **{}** )  
 // Control de transparencia  
 AlphaSlider(  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*padding*(8.*dp*),  
 controller = controller  
 )  
 // Control de brillo  
 BrightnessSlider(  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*padding*(8.*dp*),  
 controller = controller  
 )  
 **}  
 }** )  
}

package es.nuskysoftware.cajamercadillos.ui.components  
  
import androidx.compose.foundation.clickable  
import androidx.compose.foundation.layout.\*  
import androidx.compose.material.icons.Icons  
import androidx.compose.material.icons.filled.*ArrowBack*import androidx.compose.material.icons.filled.*Notifications*import androidx.compose.material3.\*  
import androidx.compose.runtime.Composable  
import androidx.compose.runtime.collectAsState  
import androidx.compose.runtime.getValue  
import androidx.compose.ui.Alignment  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.unit.dp  
import androidx.navigation.NavController  
import es.nuskysoftware.cajamercadillos.ui.viewmodel.AvisoViewModel  
  
*/\*\*  
 \* HeaderBar:  
 \* ✅ Barra superior reutilizable:  
 \* - Flecha atrás opcional.  
 \* - Título dinámico.  
 \* - Icono de campana con Badge dinámico.  
 \*/*@OptIn(ExperimentalMaterial3Api::class)  
@Composable  
fun HeaderBar(  
 titulo: String,  
 avisoViewModel: AvisoViewModel,  
 navController: NavController,  
 mostrarBack: Boolean = false,  
 onClickCampana: () -> Unit  
) {  
 val avisosNoLeidos by avisoViewModel.avisosNoLeidosCount.collectAsState()  
  
 TopAppBar(  
 title = **{** Text(  
 text = titulo,  
 style = MaterialTheme.typography.titleLarge  
 )  
 **}**,  
 navigationIcon = **{** if (mostrarBack) {  
 IconButton(onClick = **{** navController.popBackStack() **}**) **{** Icon(  
 imageVector = Icons.Default.*ArrowBack*,  
 contentDescription = "Volver"  
 )  
 **}** }  
 **}**,  
 actions = **{** BadgedBox(  
 badge = **{** if (avisosNoLeidos > 0) {  
 Badge **{** Text(avisosNoLeidos.toString())  
 **}** }  
 **}** ) **{** IconButton(onClick = onClickCampana) **{** Icon(  
 imageVector = Icons.Default.*Notifications*,  
 contentDescription = "Avisos"  
 )  
 **}  
 }  
 }** )  
}

package es.nuskysoftware.cajamercadillos.ui.components  
  
import android.annotation.SuppressLint  
import androidx.compose.animation.AnimatedContent  
import androidx.compose.animation.ExperimentalAnimationApi  
import androidx.compose.animation.core.tween  
import androidx.compose.animation.fadeIn  
import androidx.compose.animation.fadeOut  
import androidx.compose.animation.togetherWith  
import androidx.compose.runtime.Composable  
import androidx.compose.ui.Modifier  
  
*/\*\*  
 \* ResponsiveAnimatedContainer:  
 \* Esta función envuelve contenido y anima los cambios  
 \* cuando el "key" cambia (por ejemplo, orientación).  
 \*  
 \* Parámetros:  
 \* - key: cualquier valor que cambie (ej. true/false para horizontal)  
 \* - modifier: modificadores opcionales para ajustar tamaño/estilo  
 \* - animationDuration: duración de la animación (ms)  
 \* - content: contenido composable que queremos animar  
 \*  
 \* Uso:  
 \* ResponsiveAnimatedContainer(key = isHorizontal) {  
 \* if (isHorizontal) Row { ... } else Column { ... }  
 \* }  
 \*/*@SuppressLint("UnusedContentLambdaTargetStateParameter")  
@OptIn(ExperimentalAnimationApi::class)  
@Composable  
fun ResponsiveAnimatedContainer(  
 key: Any, // Estado que dispara la animación (ej. orientación)  
 modifier: Modifier = Modifier,  
 animationDuration: Int = 400, // Duración animación en ms  
 content: @Composable () -> Unit  
) {  
 AnimatedContent(  
 targetState = key,  
 transitionSpec = **{** *fadeIn*(*tween*(animationDuration)) *togetherWith fadeOut*(*tween*(animationDuration))  
 **}**,  
 modifier = modifier,  
 label = "ResponsiveAnimation"  
 ) **{** content()  
 **}**}

package es.nuskysoftware.cajamercadillos.ui.pantallas  
  
import android.content.res.Configuration  
import androidx.compose.foundation.layout.\*  
import androidx.compose.material3.MaterialTheme  
import androidx.compose.runtime.Composable  
import androidx.compose.ui.Alignment  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.platform.*LocalConfiguration*import androidx.compose.ui.unit.dp  
import androidx.navigation.NavController  
import es.nuskysoftware.cajamercadillos.R  
import es.nuskysoftware.cajamercadillos.ui.components.BotonHome  
import es.nuskysoftware.cajamercadillos.ui.components.HeaderBar  
import es.nuskysoftware.cajamercadillos.ui.components.ResponsiveAnimatedContainer  
import es.nuskysoftware.cajamercadillos.utils.FooterMarca  
import es.nuskysoftware.cajamercadillos.ui.viewmodel.AvisoViewModel  
  
*/\*\*  
 \* PantallaArticulosCategorias:  
 \* ✅ Usa HeaderBar con badge dinámico y botón back.  
 \* ✅ Botones principales responsive.  
 \* ✅ Footer © La Nusky Software.  
 \*/*@Composable  
fun PantallaArticulosCategorias(  
 navController: NavController,  
 avisoViewModel: AvisoViewModel  
) {  
 val configuracion = *LocalConfiguration*.current  
 val isHorizontal = configuracion.orientation == Configuration.*ORIENTATION\_LANDSCAPE* Column(  
 modifier = Modifier  
 .*fillMaxSize*()  
 .*statusBarsPadding*()  
 .*padding*(16.*dp*),  
 verticalArrangement = Arrangement.SpaceBetween,  
 horizontalAlignment = Alignment.CenterHorizontally  
 ) **{** // ---------- HEADER ----------  
 HeaderBar(  
 titulo = "Artículos y Categorías",  
 avisoViewModel = avisoViewModel,  
 navController = navController,  
 mostrarBack = true,  
 onClickCampana = **{** navController.navigate("pantalla\_avisos")  
 **}** )  
  
 Spacer(modifier = Modifier.*height*(24.*dp*))  
  
 // ---------- BOTONES PRINCIPALES (Responsive con animación) ----------  
 ResponsiveAnimatedContainer(key = isHorizontal) **{** if (isHorizontal) {  
 // Landscape → Botones en fila  
 Row(  
 modifier = Modifier.*fillMaxWidth*(),  
 horizontalArrangement = Arrangement.SpaceEvenly  
 ) **{** BotonHome(  
 iconRes = R.drawable.*articulos\_home*,  
 texto = "Artículos",  
 colorFondo = MaterialTheme.colorScheme.secondary,  
 colorTexto = MaterialTheme.colorScheme.onPrimary  
 ) **{** // *TODO: navController.navigate("pantallaArticulos")* **}** BotonHome(  
 iconRes = R.drawable.*categorias\_home*,  
 texto = "Categorías",  
 colorFondo = MaterialTheme.colorScheme.primary,  
 colorTexto = MaterialTheme.colorScheme.onPrimary  
 ) **{** navController.navigate("categorias")  
 **}  
 }** } else {  
 // Portrait → Botones en columna  
 Column(  
 verticalArrangement = Arrangement.spacedBy(16.*dp*),  
 horizontalAlignment = Alignment.CenterHorizontally  
 ) **{** BotonHome(  
 iconRes = R.drawable.*articulos\_home*,  
 texto = "Artículos",  
 colorFondo = MaterialTheme.colorScheme.secondary,  
 colorTexto = MaterialTheme.colorScheme.onPrimary  
 ) **{** // *TODO: navController.navigate("pantallaArticulos")* **}** BotonHome(  
 iconRes = R.drawable.*categorias\_home*,  
 texto = "Categorías",  
 colorFondo = MaterialTheme.colorScheme.primary,  
 colorTexto = MaterialTheme.colorScheme.onPrimary  
 ) **{** navController.navigate("categorias")  
 **}  
 }** }  
 **}** Spacer(modifier = Modifier.*weight*(1f))  
  
 // ---------- FOOTER ----------  
 FooterMarca()  
 **}**}

package es.nuskysoftware.cajamercadillos.ui.pantallas  
  
import androidx.compose.foundation.clickable  
import androidx.compose.foundation.layout.\*  
import androidx.compose.foundation.lazy.LazyColumn  
import androidx.compose.foundation.lazy.items  
import androidx.compose.ui.res.painterResource  
import es.nuskysoftware.cajamercadillos.R  
import androidx.compose.material3.\*  
import androidx.compose.runtime.\*  
import androidx.compose.ui.Alignment  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.unit.dp  
import androidx.lifecycle.viewmodel.compose.viewModel  
import androidx.navigation.NavController  
import es.nuskysoftware.cajamercadillos.data.local.entity.AvisoConDetalle  
import es.nuskysoftware.cajamercadillos.ui.viewmodel.AvisoViewModel  
import kotlinx.coroutines.launch  
import java.text.SimpleDateFormat  
import java.util.\*  
  
@OptIn(ExperimentalMaterial3Api::class)  
@Composable  
fun PantallaAvisos(  
 navController: NavController,  
 viewModel: AvisoViewModel = viewModel()  
) {  
 val avisos by viewModel.avisosConDetalle.collectAsState()  
  
 val sheetState = rememberModalBottomSheetState(skipPartiallyExpanded = true)  
 val scope = rememberCoroutineScope()  
  
 var avisoSeleccionado by remember **{** *mutableStateOf*<AvisoConDetalle?>(null) **}** Scaffold(  
 topBar = **{** TopAppBar(  
 title = **{** Text(  
 text = "Avisos",  
 style = MaterialTheme.typography.titleLarge  
 )  
 **}**,  
 navigationIcon = **{** IconButton(onClick = **{** navController.popBackStack() **}**) **{** Icon(  
 painter = painterResource(id = R.drawable.*arrow\_back*),  
 contentDescription = "Volver",  
 tint = MaterialTheme.colorScheme.onSurface  
 )  
 **}  
 }** )  
 **}** ) **{** padding **->** Column(  
 modifier = Modifier  
 .*fillMaxSize*()  
 .*padding*(padding)  
 ) **{** if (avisos.isEmpty()) {  
 Box(  
 modifier = Modifier.*fillMaxSize*(),  
 contentAlignment = Alignment.Center  
 ) **{** Text(  
 text = "No hay avisos por el momento.",  
 style = MaterialTheme.typography.bodyMedium,  
 color = MaterialTheme.colorScheme.onSurface.copy(alpha = 0.7f)  
 )  
 **}** } else {  
 LazyColumn(  
 modifier = Modifier.*fillMaxSize*(),  
 contentPadding = *PaddingValues*(16.*dp*),  
 verticalArrangement = Arrangement.spacedBy(12.*dp*)  
 ) **{** *items*(avisos) **{** aviso **->** AvisoItem(  
 aviso = aviso,  
 onClick = **{** avisoSeleccionado = aviso  
 scope.*launch* **{** sheetState.show() **}  
 }** )  
 **}  
 }** }  
 **}  
 }** if (avisoSeleccionado != null) {  
 ModalBottomSheet(  
 onDismissRequest = **{** scope.*launch* **{** sheetState.hide()  
 avisoSeleccionado = null  
 **}  
 }**,  
 sheetState = sheetState,  
 containerColor = MaterialTheme.colorScheme.surface  
 ) **{** avisoSeleccionado?.*let* **{** aviso **->** DetalleAvisoSheet(  
 aviso = aviso,  
 onMarcarLeido = **{** viewModel.marcarAvisoComoLeido(aviso.avisoUsuario.idAvisoUsuario)  
 scope.*launch* **{** sheetState.hide()  
 avisoSeleccionado = null  
 **}  
 }** )  
 **}  
 }** }  
}  
  
@Composable  
fun AvisoItem(  
 aviso: AvisoConDetalle,  
 onClick: () -> Unit  
) {  
 Card(  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*clickable* **{** onClick() **}**,  
 elevation = CardDefaults.cardElevation(defaultElevation = 4.*dp*),  
 colors = CardDefaults.cardColors(  
 containerColor = MaterialTheme.colorScheme.surfaceVariant  
 )  
 ) **{** Row(  
 modifier = Modifier  
 .*padding*(16.*dp*)  
 .*fillMaxWidth*(),  
 horizontalArrangement = Arrangement.SpaceBetween,  
 verticalAlignment = Alignment.CenterVertically  
 ) **{** Column **{** Text(  
 text = aviso.avisoBase.titulo,  
 style = MaterialTheme.typography.titleMedium,  
 color = MaterialTheme.colorScheme.onSurface  
 )  
 Spacer(modifier = Modifier.*height*(4.*dp*))  
 Text(  
 text = *formatearFecha*(aviso.avisoUsuario.fechaCreacion),  
 style = MaterialTheme.typography.bodySmall,  
 color = MaterialTheme.colorScheme.onSurface.copy(alpha = 0.7f)  
 )  
 **}** if (!aviso.avisoUsuario.leido) {  
 AssistChip(  
 onClick = **{}**,  
 label = **{** Text("Nuevo", style = MaterialTheme.typography.bodySmall)  
 **}**,  
 colors = AssistChipDefaults.assistChipColors(  
 containerColor = MaterialTheme.colorScheme.primaryContainer,  
 labelColor = MaterialTheme.colorScheme.onPrimaryContainer  
 )  
 )  
 }  
 **}  
 }**}  
  
@Composable  
fun DetalleAvisoSheet(  
 aviso: AvisoConDetalle,  
 onMarcarLeido: () -> Unit  
) {  
 Column(  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*padding*(24.*dp*)  
 ) **{** Text(  
 text = aviso.avisoBase.titulo,  
 style = MaterialTheme.typography.titleLarge,  
 color = MaterialTheme.colorScheme.onSurface  
 )  
 Spacer(modifier = Modifier.*height*(12.*dp*))  
 Text(  
 text = aviso.avisoBase.mensaje,  
 style = MaterialTheme.typography.bodyMedium,  
 color = MaterialTheme.colorScheme.onSurface.copy(alpha = 0.9f)  
 )  
 Spacer(modifier = Modifier.*height*(16.*dp*))  
 Text(  
 text = "Fecha: ${*formatearFecha*(aviso.avisoUsuario.fechaCreacion)}",  
 style = MaterialTheme.typography.bodySmall,  
 color = MaterialTheme.colorScheme.onSurface.copy(alpha = 0.6f)  
 )  
 Spacer(modifier = Modifier.*height*(24.*dp*))  
  
 if (!aviso.avisoUsuario.leido) {  
 Button(  
 onClick = onMarcarLeido,  
 modifier = Modifier.*fillMaxWidth*(),  
 colors = ButtonDefaults.buttonColors(  
 containerColor = MaterialTheme.colorScheme.primary,  
 contentColor = MaterialTheme.colorScheme.onPrimary  
 )  
 ) **{** Text("Marcar como leído")  
 **}** }  
 **}**}  
  
fun formatearFecha(timestamp: Long): String {  
 val sdf = SimpleDateFormat("dd/MM/yyyy HH:mm", Locale.getDefault())  
 return sdf.format(Date(timestamp))  
}

package es.nuskysoftware.cajamercadillos.ui.pantallas  
  
import android.content.res.Configuration  
import androidx.compose.animation.AnimatedVisibility  
import androidx.compose.foundation.background  
import androidx.compose.foundation.clickable  
import androidx.compose.foundation.layout.\*  
import androidx.compose.foundation.lazy.LazyColumn  
import androidx.compose.foundation.lazy.items  
import androidx.compose.foundation.shape.*CircleShape*import androidx.compose.foundation.shape.RoundedCornerShape  
import androidx.compose.material3.\*  
import androidx.compose.runtime.\*  
import androidx.compose.ui.Alignment  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.draw.clip  
import androidx.compose.ui.graphics.Color  
import androidx.compose.ui.graphics.toArgb  
import androidx.compose.ui.platform.*LocalConfiguration*import androidx.compose.ui.res.painterResource  
import androidx.compose.ui.text.style.TextAlign  
import androidx.compose.ui.unit.dp  
import androidx.navigation.NavController  
import es.nuskysoftware.cajamercadillos.R  
import es.nuskysoftware.cajamercadillos.data.local.entity.CategoriaEntity  
import es.nuskysoftware.cajamercadillos.ui.components.DialogSelectorColor  
import es.nuskysoftware.cajamercadillos.ui.components.CustomDialogConfirmacion  
import es.nuskysoftware.cajamercadillos.ui.components.HeaderBar  
import es.nuskysoftware.cajamercadillos.ui.components.ResponsiveAnimatedContainer  
import es.nuskysoftware.cajamercadillos.ui.viewmodel.CategoriaViewModel  
import es.nuskysoftware.cajamercadillos.ui.viewmodel.AvisoViewModel  
import es.nuskysoftware.cajamercadillos.utils.FooterMarca  
import es.nuskysoftware.cajamercadillos.utils.generarColorAleatorioPastel  
import kotlinx.coroutines.launch  
  
@Composable  
fun PantallaCategorias(  
 navController: NavController,  
 viewModel: CategoriaViewModel,  
 avisoViewModel: AvisoViewModel  
) {  
 val categorias by viewModel.categorias.collectAsState()  
  
 var mostrarFormulario by remember **{** *mutableStateOf*(false) **}** var nombreCategoria by remember **{** *mutableStateOf*("") **}** var colorSeleccionado by remember **{** *mutableStateOf*(*Color*(0xFFD1C4E9)) **}** var modoEdicion by remember **{** *mutableStateOf*<CategoriaEntity?>(null) **}** var mostrarColorPicker by remember **{** *mutableStateOf*(false) **}** var mostrarDialogoEliminar by remember **{** *mutableStateOf*(false) **}** var categoriaAEliminar by remember **{** *mutableStateOf*<CategoriaEntity?>(null) **}** val snackbarHostState = remember **{** SnackbarHostState() **}** val scope = rememberCoroutineScope()  
  
 val configuracion = *LocalConfiguration*.current  
 val isHorizontal = configuracion.orientation == Configuration.*ORIENTATION\_LANDSCAPE* Scaffold(snackbarHost = **{** SnackbarHost(snackbarHostState) **}**) **{** padding **->** Column(  
 modifier = Modifier  
 .*fillMaxSize*()  
 .*background*(MaterialTheme.colorScheme.background)  
 .*padding*(padding)  
 .*statusBarsPadding*()  
 ) **{** // ---------- HEADER BAR ----------  
 HeaderBar(  
 titulo = "Categorías",  
 avisoViewModel = avisoViewModel,  
 navController = navController,  
 mostrarBack = true,  
 onClickCampana = **{** navController.navigate("pantalla\_avisos")  
 **}** )  
  
 Spacer(Modifier.*height*(12.*dp*))  
  
 // ---------- FORMULARIO ----------  
 AnimatedVisibility(visible = mostrarFormulario) **{** Column(  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*background*(MaterialTheme.colorScheme.secondary.copy(alpha = 0.2f), *RoundedCornerShape*(12.*dp*))  
 .*padding*(16.*dp*)  
 ) **{** Text(  
 if (modoEdicion == null) "Nueva Categoría" else "Editar Categoría",  
 style = MaterialTheme.typography.titleMedium  
 )  
 Spacer(Modifier.*height*(8.*dp*))  
  
 OutlinedTextField(  
 value = nombreCategoria,  
 onValueChange = **{** nombreCategoria = **it }**,  
 label = **{** Text("Nombre") **}**,  
 modifier = Modifier.*fillMaxWidth*()  
 )  
  
 Spacer(Modifier.*height*(8.*dp*))  
  
 Row(verticalAlignment = Alignment.CenterVertically, modifier = Modifier.*clickable* **{** mostrarColorPicker = true **}**) **{** Text("Seleccionar color", color = MaterialTheme.colorScheme.onSurface)  
 Spacer(Modifier.*width*(12.*dp*))  
 Box(  
 modifier = Modifier  
 .*size*(32.*dp*)  
 .*clip*(*CircleShape*)  
 .*background*(colorSeleccionado)  
 )  
 **}** Spacer(Modifier.*height*(12.*dp*))  
  
 Row(horizontalArrangement = Arrangement.spacedBy(12.*dp*)) **{** Button(onClick = **{** if (nombreCategoria.*isBlank*()) {  
 scope.*launch* **{** snackbarHostState.showSnackbar("El nombre no puede estar vacío") **}** } else {  
 val colorFinal = if (colorSeleccionado == *Color*(0xFFD1C4E9)) {  
 *generarColorAleatorioPastel*()  
 } else colorSeleccionado  
  
 val colorHex = String.*format*("#%06X", 0xFFFFFF and colorFinal.*toArgb*())  
  
 if (modoEdicion == null) {  
 viewModel.insertarCategoria(nombreCategoria, colorHex)  
 } else {  
 val categoriaEditada = modoEdicion!!.copy(nombre = nombreCategoria, colorHex = colorHex)  
 viewModel.actualizarCategoria(categoriaEditada)  
 }  
  
 nombreCategoria = ""  
 colorSeleccionado = *Color*(0xFFD1C4E9)  
 modoEdicion = null  
 mostrarFormulario = false  
 }  
 **}**) **{** Text("Guardar") **}** OutlinedButton(onClick = **{** mostrarFormulario = false  
 nombreCategoria = ""  
 colorSeleccionado = *Color*(0xFFD1C4E9)  
 modoEdicion = null  
 **}**) **{** Text("Cancelar") **}  
 }  
 }  
 }** Spacer(Modifier.*height*(16.*dp*))  
  
 // ---------- LISTADO ----------  
 ResponsiveAnimatedContainer(key = categorias.size) **{** if (categorias.isEmpty()) {  
 Text("No hay categorías aún", textAlign = TextAlign.Center, modifier = Modifier.*fillMaxWidth*())  
 } else {  
 LazyColumn(modifier = Modifier.*weight*(1f)) **{** items(categorias) **{** categoria **->** Row(  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*padding*(vertical = 4.*dp*)  
 .*background*(MaterialTheme.colorScheme.surface, *RoundedCornerShape*(8.*dp*))  
 .*padding*(12.*dp*),  
 verticalAlignment = Alignment.CenterVertically,  
 horizontalArrangement = Arrangement.SpaceBetween  
 ) **{** Row(verticalAlignment = Alignment.CenterVertically) **{** Box(  
 modifier = Modifier  
 .*size*(20.*dp*)  
 .*clip*(*CircleShape*)  
 .*background*(*Color*(android.graphics.Color.parseColor(categoria.colorHex)))  
 )  
 Spacer(Modifier.*width*(8.*dp*))  
 Text(categoria.nombre, style = MaterialTheme.typography.bodyLarge)  
 **}** Row **{** IconButton(onClick = **{** mostrarFormulario = true  
 nombreCategoria = categoria.nombre  
 colorSeleccionado = *Color*(android.graphics.Color.parseColor(categoria.colorHex))  
 modoEdicion = categoria  
 **}**) **{** Icon(painter = painterResource(id = R.drawable.*edit\_icono*), contentDescription = "Editar")  
 **}** IconButton(onClick = **{** categoriaAEliminar = categoria  
 mostrarDialogoEliminar = true  
 **}**) **{** Icon(painter = painterResource(id = R.drawable.*delete\_icono*), contentDescription = "Eliminar")  
 **}  
 }  
 }  
 }  
 }** }  
 **}** Spacer(modifier = Modifier.*weight*(1f))  
 FooterMarca()  
 **}** // ---------- FAB (Botón añadir) ----------  
 Box(  
 modifier = Modifier  
 .*fillMaxSize*()  
 .*padding*(end = 24.*dp*, bottom = 80.*dp*),  
 contentAlignment = Alignment.BottomEnd  
 ) **{** if (!mostrarFormulario) {  
 FloatingActionButton(  
 onClick = **{** mostrarFormulario = true  
 nombreCategoria = ""  
 colorSeleccionado = *Color*(0xFFD1C4E9)  
 modoEdicion = null  
 **}**,  
 shape = *CircleShape*,  
 containerColor = MaterialTheme.colorScheme.primary,  
 contentColor = MaterialTheme.colorScheme.onPrimary,  
 elevation = FloatingActionButtonDefaults.elevation(8.*dp*)  
 ) **{** Icon(  
 painter = painterResource(id = R.drawable.*add\_icono*),  
 contentDescription = "Añadir categoría",  
 modifier = Modifier.*size*(28.*dp*)  
 )  
 **}** }  
 **}** // ---------- DIÁLOGOS ----------  
 if (mostrarColorPicker) {  
 DialogSelectorColor(  
 onColorElegido = **{** color **->** colorSeleccionado = color  
 mostrarColorPicker = false  
 **}**,  
 onCancelar = **{** mostrarColorPicker = false **}** )  
 }  
  
 if (mostrarDialogoEliminar && categoriaAEliminar != null) {  
 CustomDialogConfirmacion(  
 titulo = "Eliminar Categoría",  
 mensaje = "¿Seguro que deseas eliminar \"${categoriaAEliminar!!.nombre}\"?",  
 textoConfirmar = "Eliminar",  
 colorConfirmar = MaterialTheme.colorScheme.error,  
 onConfirmar = **{** viewModel.eliminarCategoria(categoriaAEliminar!!)  
 mostrarDialogoEliminar = false  
 **}**,  
 onCancelar = **{** mostrarDialogoEliminar = false **}** )  
 }  
 **}**}

package es.nuskysoftware.cajamercadillos.ui.pantallas  
  
import androidx.compose.animation.core.Animatable  
import androidx.compose.animation.core.*FastOutSlowInEasing*import androidx.compose.animation.core.tween  
import androidx.compose.foundation.Image  
import androidx.compose.foundation.background  
import androidx.compose.foundation.layout.\*  
import androidx.compose.foundation.rememberScrollState  
import androidx.compose.foundation.verticalScroll  
import androidx.compose.material3.\*  
import androidx.compose.runtime.\*  
import androidx.compose.ui.Alignment  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.draw.alpha  
import androidx.compose.ui.graphics.Color  
import androidx.compose.ui.res.painterResource  
import androidx.compose.ui.text.font.FontWeight  
import androidx.compose.ui.unit.dp  
import androidx.compose.ui.unit.sp  
import androidx.navigation.NavController  
import es.nuskysoftware.cajamercadillos.R  
import es.nuskysoftware.cajamercadillos.data.local.entity.ConfiguracionEntity  
import es.nuskysoftware.cajamercadillos.ui.viewmodel.ConfiguracionViewModel  
import kotlinx.coroutines.launch  
  
@Composable  
fun PantallaConfiguracion(  
 navController: NavController,  
 viewModel: ConfiguracionViewModel,  
 versionApp: String = "FREE"  
) {  
 val alphaAnim = remember **{** *Animatable*(0f) **}** LaunchedEffect(Unit) **{** alphaAnim.animateTo(1f, animationSpec = *tween*(1000, easing = *FastOutSlowInEasing*))  
 **}** val configuracionActual by viewModel.configuracion.collectAsState()  
  
 var idioma by remember **{** *mutableStateOf*(configuracionActual?.idioma ?: "es") **}** var moneda by remember **{** *mutableStateOf*(configuracionActual?.moneda ?: "€ Euro") **}** var fuente by remember **{** *mutableStateOf*(configuracionActual?.fuente ?: "Montserrat") **}** var temaOscuro by remember **{** *mutableStateOf*(configuracionActual?.temaOscuro ?: false) **}** var controlStock by remember **{** *mutableStateOf*((configuracionActual?.controlStock ?: 0) == 1) **}** var controlCoste by remember **{** *mutableStateOf*((configuracionActual?.controlCoste ?: 0) == 1) **}** var showIdiomaMenu by remember **{** *mutableStateOf*(false) **}** var showMonedaMenu by remember **{** *mutableStateOf*(false) **}** var showFuenteMenu by remember **{** *mutableStateOf*(false) **}** val scope = rememberCoroutineScope()  
  
 Column(  
 modifier = Modifier  
 .*fillMaxSize*()  
 .*background*(MaterialTheme.colorScheme.primary.copy(alpha = 0.05f))  
 .*verticalScroll*(rememberScrollState())  
 .*padding*(16.*dp*),  
 horizontalAlignment = Alignment.CenterHorizontally  
 ) **{** Image(  
 painter = painterResource(id = R.drawable.*splash\_logo*),  
 contentDescription = "Logo Caja Mercadillos",  
 modifier = Modifier  
 .*size*(120.*dp*)  
 .*alpha*(alphaAnim.value)  
 )  
 Spacer(Modifier.*height*(16.*dp*))  
  
 Text(  
 text = "Configuración inicial",  
 fontSize = 22.*sp*,  
 fontWeight = FontWeight.Bold,  
 color = MaterialTheme.colorScheme.onBackground,  
 modifier = Modifier.*alpha*(alphaAnim.value)  
 )  
 Spacer(Modifier.*height*(24.*dp*))  
  
 Card(modifier = Modifier.*fillMaxWidth*(), elevation = CardDefaults.cardElevation(4.*dp*)) **{** Column(Modifier.*padding*(16.*dp*)) **{** Text("Opciones Básicas", fontWeight = FontWeight.Bold, fontSize = 18.*sp*)  
 Spacer(Modifier.*height*(16.*dp*))  
  
 DropdownOption(  
 label = "Idioma",  
 selectedValue = if (idioma == "es") "Español" else "Inglés",  
 options = *listOf*("Español", "Inglés"),  
 expanded = showIdiomaMenu,  
 onExpandChange = **{** showIdiomaMenu = **it }**,  
 onSelect = **{** idioma = if (**it** == "Español") "es" else "en"  
 showIdiomaMenu = false  
 **}** )  
  
 Spacer(Modifier.*height*(16.*dp*))  
  
 DropdownOption(  
 label = "Moneda",  
 selectedValue = moneda,  
 options = *listOf*("€ Euro", "$ USD", "£ GBP"),  
 expanded = showMonedaMenu,  
 onExpandChange = **{** showMonedaMenu = **it }**,  
 onSelect = **{** moneda = **it**; showMonedaMenu = false **}** )  
  
 Spacer(Modifier.*height*(16.*dp*))  
  
 DropdownOption(  
 label = "Fuente",  
 selectedValue = fuente,  
 options = *listOf*("Montserrat", "Roboto", "Sans Serif", "Serif"),  
 expanded = showFuenteMenu,  
 onExpandChange = **{** showFuenteMenu = **it }**,  
 onSelect = **{** fuente = **it**; showFuenteMenu = false **}** )  
  
 Spacer(Modifier.*height*(16.*dp*))  
  
 Row(  
 Modifier.*fillMaxWidth*(),  
 horizontalArrangement = Arrangement.SpaceBetween,  
 verticalAlignment = Alignment.CenterVertically  
 ) **{** Text("Tema oscuro", fontWeight = FontWeight.Medium)  
 Switch(checked = temaOscuro, onCheckedChange = **{** temaOscuro = **it }**)  
 **}  
 }  
 }** Spacer(Modifier.*height*(24.*dp*))  
  
 Card(modifier = Modifier.*fillMaxWidth*(), elevation = CardDefaults.cardElevation(4.*dp*)) **{** Column(Modifier.*padding*(16.*dp*)) **{** Text("Funciones Premium", fontWeight = FontWeight.Bold, fontSize = 18.*sp*)  
 Spacer(Modifier.*height*(16.*dp*))  
  
 PremiumSwitch("Control de Stock", controlStock, versionApp != "FREE") **{** controlStock = **it }** PremiumSwitch("Control de Coste", controlCoste, versionApp != "FREE") **{** controlCoste = **it }  
 }  
 }** Spacer(Modifier.*height*(32.*dp*))  
  
 Button(  
 onClick = **{** scope.*launch* **{** val config = ConfiguracionEntity(  
 isConfigurated = true,  
 idioma = idioma,  
 moneda = moneda,  
 temaOscuro = temaOscuro,  
 fuente = fuente,  
 controlStock = if (controlStock) 1 else 0,  
 controlCoste = if (controlCoste) 1 else 0,  
 versionApp = versionApp  
 )  
 viewModel.guardarConfiguracion(config)  
 navController.navigate("home") **{** popUpTo("configuracion") **{** inclusive = true **}  
 }  
 }  
 }**,  
 modifier = Modifier.*fillMaxWidth*().*height*(50.*dp*)  
 ) **{** Text("GUARDAR Y CONTINUAR", fontWeight = FontWeight.Bold)  
 **}** Spacer(Modifier.*height*(16.*dp*))  
  
 OutlinedButton(onClick = **{ }**, modifier = Modifier.*fillMaxWidth*().*height*(50.*dp*)) **{** Text("MEJORAR A PREMIUM", fontWeight = FontWeight.Bold)  
 **}** Spacer(Modifier.*height*(24.*dp*))  
 Text("® La Nusky Software", color = Color.Gray, fontSize = 14.*sp*)  
 **}**}  
  
@Composable  
fun PremiumSwitch(label: String, value: Boolean, enabled: Boolean, onChange: (Boolean) -> Unit) {  
 Row(Modifier.*fillMaxWidth*(), horizontalArrangement = Arrangement.SpaceBetween, verticalAlignment = Alignment.CenterVertically) **{** Text(label)  
 if (enabled) Switch(checked = value, onCheckedChange = onChange) else Text("Solo Premium", color = Color.Gray, fontSize = 14.*sp*)  
 **}** Spacer(Modifier.*height*(8.*dp*))  
}  
  
@Composable  
fun DropdownOption(  
 label: String,  
 selectedValue: String,  
 options: List<String>,  
 expanded: Boolean,  
 onExpandChange: (Boolean) -> Unit,  
 onSelect: (String) -> Unit  
) {  
 Text(label, fontWeight = FontWeight.Medium)  
 Box **{** OutlinedButton(onClick = **{** onExpandChange(true) **}**, Modifier.*fillMaxWidth*()) **{** Text(selectedValue)  
 **}** DropdownMenu(expanded = expanded, onDismissRequest = **{** onExpandChange(false) **}**) **{** options.*forEach* **{** option **->** DropdownMenuItem(text = **{** Text(option) **}**, onClick = **{** onSelect(option) **}**)  
 **}  
 }  
 }**}

package es.nuskysoftware.cajamercadillos.ui.pantallas  
  
import android.content.res.Configuration  
import androidx.compose.foundation.clickable  
import androidx.compose.foundation.layout.\*  
import androidx.compose.foundation.lazy.LazyColumn  
import androidx.compose.foundation.lazy.items  
import androidx.compose.material.icons.Icons  
import androidx.compose.material.icons.filled.*Close*import androidx.compose.material3.\*  
import androidx.compose.runtime.\*  
import androidx.compose.ui.Alignment  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.platform.*LocalConfiguration*import androidx.compose.ui.unit.dp  
import androidx.navigation.NavController  
import es.nuskysoftware.cajamercadillos.R  
import es.nuskysoftware.cajamercadillos.data.local.entity.AvisoConDetalle  
import es.nuskysoftware.cajamercadillos.ui.components.BotonHome  
import es.nuskysoftware.cajamercadillos.ui.components.HeaderBar  
import es.nuskysoftware.cajamercadillos.ui.components.ResponsiveAnimatedContainer  
import es.nuskysoftware.cajamercadillos.ui.theme.AppTheme  
import es.nuskysoftware.cajamercadillos.utils.FooterMarca  
import es.nuskysoftware.cajamercadillos.ui.viewmodel.AvisoViewModel  
import kotlinx.coroutines.launch  
import java.text.SimpleDateFormat  
import java.util.\*  
  
*/\*\*  
 \* Pantalla principal (Home):  
 \* ✅ Incluye Header reutilizable con badge dinámico.  
 \* ✅ Botones principales (responsive con animación).  
 \* ✅ Footer con © La Nusky Software.  
 \* ✅ ModalBottomSheet para avisos.  
 \*/*@OptIn(ExperimentalMaterial3Api::class)  
@Composable  
fun PantallaHome(  
 navController: NavController,  
 avisoViewModel: AvisoViewModel  
) {  
 val configuracion = *LocalConfiguration*.current  
 val isHorizontal = configuracion.orientation == Configuration.*ORIENTATION\_LANDSCAPE* // ✅ ESTADO PARA EL MODAL DE AVISOS  
 var mostrarModalAvisos by remember **{** *mutableStateOf*(false) **}** val sheetState = rememberModalBottomSheetState(skipPartiallyExpanded = true)  
 val scope = rememberCoroutineScope()  
  
 // ✅ DATOS DE AVISOS  
 val avisos by avisoViewModel.avisosConDetalle.collectAsState()  
 var avisoSeleccionado by remember **{** *mutableStateOf*<AvisoConDetalle?>(null) **}** Column(  
 modifier = Modifier  
 .*fillMaxSize*()  
 .*statusBarsPadding*()  
 ) **{** // ---------- HEADER REUTILIZABLE ----------  
 HeaderBar(  
 titulo = "Caja Mercadillos",  
 avisoViewModel = avisoViewModel,  
 navController = navController,  
 mostrarBack = false,  
 onClickCampana = **{** // ✅ CAMBIO: Abrir modal en lugar de navegar  
 mostrarModalAvisos = true  
 **}** )  
  
 Spacer(modifier = Modifier.*height*(16.*dp*))  
  
 // ---------- BLOQUE DE BOTONES CENTRADO ----------  
 Box(  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*weight*(1f), // Ocupa *todo el espacio disponible entre Header y Footer* contentAlignment = Alignment.Center  
 ) **{** ResponsiveAnimatedContainer(key = isHorizontal) **{** if (isHorizontal) {  
 Row(  
 horizontalArrangement = Arrangement.SpaceEvenly,  
 verticalAlignment = Alignment.CenterVertically  
 ) **{** BotonHome(  
 iconRes = R.drawable.*mercadillo\_home*,  
 texto = "Mercadillo",  
 colorFondo = MaterialTheme.colorScheme.primary,  
 colorTexto = MaterialTheme.colorScheme.onPrimary  
 ) **{ }** BotonHome(  
 iconRes = R.drawable.*articulos\_home*,  
 texto = "Artículos\nCategorías",  
 colorFondo = MaterialTheme.colorScheme.secondary,  
 colorTexto = MaterialTheme.colorScheme.onPrimary  
 ) **{** navController.navigate("articulos\_categorias")  
 **}** BotonHome(  
 iconRes = R.drawable.*listados\_home*,  
 texto = "Listados",  
 colorFondo = AppTheme.extendedColors.info,  
 colorTexto = MaterialTheme.colorScheme.onPrimary  
 ) **{ }** BotonHome(  
 iconRes = R.drawable.*configuracion\_home*,  
 texto = "Configuración",  
 colorFondo = AppTheme.extendedColors.warning,  
 colorTexto = MaterialTheme.colorScheme.onPrimary  
 ) **{** navController.navigate("configuracion")  
 **}  
 }** } else {  
 Column(horizontalAlignment = Alignment.CenterHorizontally) **{** Row(horizontalArrangement = Arrangement.SpaceEvenly) **{** BotonHome(  
 iconRes = R.drawable.*mercadillo\_home*,  
 texto = "Mercadillo",  
 colorFondo = MaterialTheme.colorScheme.primary,  
 colorTexto = MaterialTheme.colorScheme.onPrimary  
 ) **{ }** BotonHome(  
 iconRes = R.drawable.*articulos\_home*,  
 texto = "Artículos\nCategorías",  
 colorFondo = MaterialTheme.colorScheme.secondary,  
 colorTexto = MaterialTheme.colorScheme.onPrimary  
 ) **{** navController.navigate("articulos\_categorias")  
 **}  
 }** Spacer(modifier = Modifier.*height*(16.*dp*))  
  
 Row(horizontalArrangement = Arrangement.SpaceEvenly) **{** BotonHome(  
 iconRes = R.drawable.*listados\_home*,  
 texto = "Listados",  
 colorFondo = AppTheme.extendedColors.info,  
 colorTexto = MaterialTheme.colorScheme.onPrimary  
 ) **{ }** BotonHome(  
 iconRes = R.drawable.*configuracion\_home*,  
 texto = "Configuración",  
 colorFondo = AppTheme.extendedColors.warning,  
 colorTexto = MaterialTheme.colorScheme.onPrimary  
 ) **{** navController.navigate("configuracion")  
 **}  
 }  
 }** }  
 **}  
 }** // ---------- FOOTER ----------  
 FooterMarca()  
 **}** // ✅ MODAL BOTTOM SHEET PARA AVISOS  
 if (mostrarModalAvisos) {  
 ModalBottomSheet(  
 onDismissRequest = **{** mostrarModalAvisos = false  
 avisoSeleccionado = null  
 **}**,  
 sheetState = sheetState,  
 containerColor = MaterialTheme.colorScheme.surface  
 ) **{** ModalAvisosContent(  
 avisos = avisos,  
 onAvisoClick = **{** aviso **->** avisoSeleccionado = aviso  
 **}**,  
 onMarcarLeido = **{** aviso **->** avisoViewModel.marcarAvisoComoLeido(aviso.avisoUsuario.idAvisoUsuario)  
 avisoSeleccionado = null  
 **}**,  
 onCerrar = **{** scope.*launch* **{** sheetState.hide()  
 mostrarModalAvisos = false  
 avisoSeleccionado = null  
 **}  
 }**,  
 avisoSeleccionado = avisoSeleccionado  
 )  
 **}** }  
}  
  
// ✅ CONTENIDO DEL MODAL DE AVISOS  
@Composable  
private fun ModalAvisosContent(  
 avisos: List<AvisoConDetalle>,  
 onAvisoClick: (AvisoConDetalle) -> Unit,  
 onMarcarLeido: (AvisoConDetalle) -> Unit,  
 onCerrar: () -> Unit,  
 avisoSeleccionado: AvisoConDetalle?  
) {  
 Column(  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*padding*(16.*dp*)  
 ) **{** // ✅ HEADER DEL MODAL  
 Row(  
 modifier = Modifier.*fillMaxWidth*(),  
 horizontalArrangement = Arrangement.SpaceBetween,  
 verticalAlignment = Alignment.CenterVertically  
 ) **{** Text(  
 text = "Avisos",  
 style = MaterialTheme.typography.titleLarge,  
 color = MaterialTheme.colorScheme.onSurface  
 )  
 IconButton(onClick = onCerrar) **{** Icon(  
 imageVector = Icons.Default.*Close*,  
 contentDescription = "Cerrar",  
 tint = MaterialTheme.colorScheme.onSurface  
 )  
 **}  
 }** Spacer(modifier = Modifier.*height*(16.*dp*))  
  
 // ✅ CONTENIDO PRINCIPAL  
 if (avisoSeleccionado != null) {  
 // Mostrar detalle del aviso seleccionado  
 DetalleAvisoContent(  
 aviso = avisoSeleccionado,  
 onMarcarLeido = **{** onMarcarLeido(avisoSeleccionado) **}**,  
 onVolver = **{** onAvisoClick(avisoSeleccionado) **}** // Reset selection  
 )  
 } else {  
 // Mostrar lista de avisos  
 if (avisos.isEmpty()) {  
 Box(  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*height*(200.*dp*),  
 contentAlignment = Alignment.Center  
 ) **{** Text(  
 text = "No hay avisos por el momento.",  
 style = MaterialTheme.typography.bodyMedium,  
 color = MaterialTheme.colorScheme.onSurface.copy(alpha = 0.7f)  
 )  
 **}** } else {  
 LazyColumn(  
 modifier = Modifier.*heightIn*(max = 400.*dp*),  
 verticalArrangement = Arrangement.spacedBy(8.*dp*)  
 ) **{** items(avisos) **{** aviso **->** AvisoItemModal(  
 aviso = aviso,  
 onClick = **{** onAvisoClick(aviso) **}** )  
 **}  
 }** }  
 }  
  
 // Spacer para que el contenido no quede pegado al bottom  
 Spacer(modifier = Modifier.*height*(24.*dp*))  
 **}**}  
  
// ✅ ITEM DE AVISO EN EL MODAL  
@Composable  
private fun AvisoItemModal(  
 aviso: AvisoConDetalle,  
 onClick: () -> Unit  
) {  
 Card(  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*clickable* **{** onClick() **}**,  
 elevation = CardDefaults.cardElevation(defaultElevation = 2.*dp*),  
 colors = CardDefaults.cardColors(  
 containerColor = MaterialTheme.colorScheme.surfaceVariant  
 )  
 ) **{** Row(  
 modifier = Modifier  
 .*padding*(12.*dp*)  
 .*fillMaxWidth*(),  
 horizontalArrangement = Arrangement.SpaceBetween,  
 verticalAlignment = Alignment.CenterVertically  
 ) **{** Column(modifier = Modifier.*weight*(1f)) **{** Text(  
 text = aviso.avisoBase.titulo,  
 style = MaterialTheme.typography.titleSmall,  
 color = MaterialTheme.colorScheme.onSurface  
 )  
 Spacer(modifier = Modifier.*height*(4.*dp*))  
 Text(  
 text = *formatearFechaModal*(aviso.avisoUsuario.fechaCreacion),  
 style = MaterialTheme.typography.bodySmall,  
 color = MaterialTheme.colorScheme.onSurface.copy(alpha = 0.7f)  
 )  
 **}** if (!aviso.avisoUsuario.leido) {  
 AssistChip(  
 onClick = **{}**,  
 label = **{** Text("Nuevo", style = MaterialTheme.typography.labelSmall)  
 **}**,  
 colors = AssistChipDefaults.assistChipColors(  
 containerColor = MaterialTheme.colorScheme.primaryContainer,  
 labelColor = MaterialTheme.colorScheme.onPrimaryContainer  
 )  
 )  
 }  
 **}  
 }**}  
  
// ✅ DETALLE DEL AVISO  
@Composable  
private fun DetalleAvisoContent(  
 aviso: AvisoConDetalle,  
 onMarcarLeido: () -> Unit,  
 onVolver: () -> Unit  
) {  
 Column(  
 modifier = Modifier.*fillMaxWidth*()  
 ) **{** Text(  
 text = aviso.avisoBase.titulo,  
 style = MaterialTheme.typography.titleMedium,  
 color = MaterialTheme.colorScheme.onSurface  
 )  
 Spacer(modifier = Modifier.*height*(12.*dp*))  
 Text(  
 text = aviso.avisoBase.mensaje,  
 style = MaterialTheme.typography.bodyMedium,  
 color = MaterialTheme.colorScheme.onSurface.copy(alpha = 0.9f)  
 )  
 Spacer(modifier = Modifier.*height*(16.*dp*))  
 Text(  
 text = "Fecha: ${*formatearFechaModal*(aviso.avisoUsuario.fechaCreacion)}",  
 style = MaterialTheme.typography.bodySmall,  
 color = MaterialTheme.colorScheme.onSurface.copy(alpha = 0.6f)  
 )  
 Spacer(modifier = Modifier.*height*(24.*dp*))  
  
 Row(  
 modifier = Modifier.*fillMaxWidth*(),  
 horizontalArrangement = Arrangement.spacedBy(12.*dp*)  
 ) **{** OutlinedButton(  
 onClick = onVolver,  
 modifier = Modifier.*weight*(1f)  
 ) **{** Text("Volver")  
 **}** if (!aviso.avisoUsuario.leido) {  
 Button(  
 onClick = onMarcarLeido,  
 modifier = Modifier.*weight*(1f),  
 colors = ButtonDefaults.buttonColors(  
 containerColor = MaterialTheme.colorScheme.primary,  
 contentColor = MaterialTheme.colorScheme.onPrimary  
 )  
 ) **{** Text("Marcar como leído")  
 **}** }  
 **}  
 }**}  
  
// ✅ HELPER FUNCTION (nombre único para evitar conflictos)  
private fun formatearFechaModal(timestamp: Long): String {  
 val sdf = SimpleDateFormat("dd/MM/yyyy HH:mm", Locale.getDefault())  
 return sdf.format(Date(timestamp))  
}

package es.nuskysoftware.cajamercadillos.ui.pantallas  
  
import androidx.compose.animation.core.Animatable  
import androidx.compose.animation.core.*FastOutSlowInEasing*import androidx.compose.animation.core.tween  
import androidx.compose.foundation.Image  
import androidx.compose.foundation.background  
import androidx.compose.foundation.layout.\*  
import androidx.compose.material3.MaterialTheme  
import androidx.compose.runtime.\*  
import androidx.compose.ui.Alignment  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.draw.alpha  
import androidx.compose.ui.res.painterResource  
import androidx.navigation.NavController  
import es.nuskysoftware.cajamercadillos.R  
import es.nuskysoftware.cajamercadillos.ui.viewmodel.ConfiguracionViewModel  
import kotlinx.coroutines.delay  
  
@Composable  
fun PantallaSplash(  
 navController: NavController,  
 configuracionViewModel: ConfiguracionViewModel  
) {  
 val alphaAnim = remember **{** *Animatable*(0f) **}** // Animación logo  
 LaunchedEffect(Unit) **{** alphaAnim.animateTo(  
 targetValue = 1f,  
 animationSpec = *tween*(durationMillis = 1200, easing = *FastOutSlowInEasing*)  
 )  
 **}** // Observamos configuración en tiempo real  
 val config by configuracionViewModel.configuracion.collectAsState()  
  
 // Navegación tras delay y verificación  
 LaunchedEffect(config) **{** delay(1500)  
 if (config == null || config?.isConfigurated == false) {  
 navController.navigate("configuracion") **{** popUpTo("splash") **{** inclusive = true **}  
 }** } else {  
 navController.navigate("home") **{** popUpTo("splash") **{** inclusive = true **}  
 }** }  
 **}** // UI  
 Box(  
 modifier = Modifier  
 .*fillMaxSize*()  
 .*background*(MaterialTheme.colorScheme.primary),  
 contentAlignment = Alignment.Center  
 ) **{** Image(  
 painter = painterResource(id = R.drawable.*splash\_logo*),  
 contentDescription = "Logo Caja Mercadillos",  
 modifier = Modifier  
 .*fillMaxWidth*(0.8f)  
 .*alpha*(alphaAnim.value)  
 )  
 **}**}

package es.nuskysoftware.cajamercadillos.ui.theme  
  
import androidx.compose.ui.graphics.Color  
  
*/\*\*  
 \* Archivo: Colors.kt  
 \* ✅ Colores base (modo claro y oscuro) + paleta extra pastel.  
 \*/*// ------------------- PALETA PRINCIPAL (MODO CLARO) -------------------  
val *PrimaryLight* = *Color*(0xFFA8D5BA) // Verde menta pastel  
val *OnPrimaryLight* = *Color*(0xFF000000) // Texto sobre primario claro  
  
val *SecondaryLight* = *Color*(0xFFC9B6E4) // Lavanda pastel  
val *OnSecondaryLight* = *Color*(0xFF000000)  
  
val *BackgroundLight* = *Color*(0xFFFAFAFA) // Fondo claro  
val *OnBackgroundLight* = *Color*(0xFF333333) // Texto sobre fondo claro  
  
// ------------------- PALETA PRINCIPAL (MODO OSCURO) -------------------  
val *PrimaryDark* = *Color*(0xFF7BAE9F)  
val *OnPrimaryDark* = *Color*(0xFFFFFFFF)  
  
val *SecondaryDark* = *Color*(0xFFA995C7)  
val *OnSecondaryDark* = *Color*(0xFFFFFFFF)  
  
val *BackgroundDark* = *Color*(0xFF121212)  
val *OnBackgroundDark* = *Color*(0xFFE1E1E1)  
  
// ------------------- COLORES EXTRA -------------------  
val *Success* = *Color*(0xFFAED581) // Verde éxito pastel  
val *Warning* = *Color*(0xFFFFD54F) // Amarillo advertencia  
val *ErrorCustom* = *Color*(0xFFE57373) // Rojo error pastel  
val *Info* = *Color*(0xFF64B5F6) // Azul info pastel

package es.nuskysoftware.cajamercadillos.ui.theme  
  
import androidx.compose.runtime.Immutable  
import androidx.compose.runtime.staticCompositionLocalOf  
import androidx.compose.ui.graphics.Color  
  
*/\*\*  
 \* Archivo: CustomColors.kt  
 \* ✅ Sistema extendido para colores personalizados en la app.  
 \* Incluye tonos extra sin romper MaterialTheme.  
 \*/*@Immutable  
data class ExtendedColors(  
 val success: Color, // Confirmaciones  
 val warning: Color, // Advertencias  
 val error: Color, // Errores  
 val info: Color, // Información  
  
 // ✅ Colores para botones extra (PantallaHome)  
 val secondary: Color,  
 val onSecondary: Color,  
 val onInfo: Color,  
 val onWarning: Color  
)  
  
// ---------------------- MODO CLARO ----------------------  
val *LightExtendedColors* = ExtendedColors(  
 success = *Success*,  
 warning = *Warning*,  
 error = *ErrorCustom*,  
 info = *Info*,  
  
 secondary = *SecondaryLight*,  
 onSecondary = *OnSecondaryLight*,  
 onInfo = Color.White, // Fondo azul claro → texto blanco  
 onWarning = Color.Black // Amarillo pastel → texto negro  
)  
  
// ---------------------- MODO OSCURO ----------------------  
val *DarkExtendedColors* = ExtendedColors(  
 success = *Success*,  
 warning = *Warning*,  
 error = *ErrorCustom*,  
 info = *Info*,  
  
 secondary = *SecondaryDark*,  
 onSecondary = *OnSecondaryDark*,  
 onInfo = Color.Black, // Fondo info pastel claro → texto negro  
 onWarning = Color.Black  
)  
  
// CompositionLocal para inyectar extendedColors en toda la UI  
val *LocalExtendedColors* = *staticCompositionLocalOf* **{** *LightExtendedColors* **}**

package es.nuskysoftware.cajamercadillos.ui.theme  
  
  
import androidx.compose.foundation.shape.RoundedCornerShape  
import androidx.compose.material3.Shapes  
import androidx.compose.ui.unit.dp  
  
*/\*\*  
 \* Archivo: Shape.kt  
 \* Define las formas (bordes redondeados) que usará la app.  
 \* Esto hace que todo el diseño sea consistente y elegante.  
 \*  
 \* NOTA:  
 \* - Material 3 ya trae un sistema de shapes, pero aquí  
 \* lo personalizamos para nuestra estética.  
 \*/*// ------------------- Definición de formas -------------------  
val *Shapes* = Shapes(  
 small = *RoundedCornerShape*(4.*dp*), // Elementos pequeños (chips, etiquetas)  
 medium = *RoundedCornerShape*(8.*dp*), // Botones, tarjetas  
 large = *RoundedCornerShape*(16.*dp*) // Diálogos, paneles grandes  
)  
  
*/\*\*  
 \* Cómo se usan:  
 \* - En cualquier Composable, por ejemplo un Card:  
 \* Card(shape = MaterialTheme.shapes.medium) { ... }  
 \*  
 \* - Botones por defecto ya usan estas formas cuando  
 \* el tema está aplicado.  
 \*/*

package es.nuskysoftware.cajamercadillos.ui.theme  
  
import androidx.compose.foundation.isSystemInDarkTheme  
import androidx.compose.material3.MaterialTheme  
import androidx.compose.material3.darkColorScheme  
import androidx.compose.material3.lightColorScheme  
import androidx.compose.runtime.Composable  
import androidx.compose.runtime.CompositionLocalProvider  
  
*/\*\*  
 \* Archivo: Theme.kt  
 \* ✅ Integra:  
 \* - Paleta Material3  
 \* - ExtendedColors  
 \* - Tipografía y Shapes  
 \*/*// ------------------- Paleta Material3 -------------------  
private val *LightColors* = *lightColorScheme*(  
 primary = *PrimaryLight*,  
 onPrimary = *OnPrimaryLight*,  
 secondary = *SecondaryLight*,  
 onSecondary = *OnSecondaryLight*,  
 background = *BackgroundLight*,  
 onBackground = *OnBackgroundLight*)  
  
private val *DarkColors* = *darkColorScheme*(  
 primary = *PrimaryDark*,  
 onPrimary = *OnPrimaryDark*,  
 secondary = *SecondaryDark*,  
 onSecondary = *OnSecondaryDark*,  
 background = *BackgroundDark*,  
 onBackground = *OnBackgroundDark*)  
  
// ------------------- Tema principal -------------------  
@Composable  
fun CajaMercadillosTheme(  
 darkTheme: Boolean = isSystemInDarkTheme(),  
 content: @Composable () -> Unit  
) {  
 val colors = if (darkTheme) *DarkColors* else *LightColors* val extendedColors = if (darkTheme) *DarkExtendedColors* else *LightExtendedColors* CompositionLocalProvider(*LocalExtendedColors* provides extendedColors) **{** MaterialTheme(  
 colorScheme = colors,  
 typography = *Typography*,  
 shapes = *Shapes*,  
 content = content  
 )  
 **}**}  
  
// ------------------- Acceso rápido a extendedColors -------------------  
object AppTheme {  
 val extendedColors: ExtendedColors  
 @Composable  
 get() = *LocalExtendedColors*.current  
}

package es.nuskysoftware.cajamercadillos.ui.theme  
  
import androidx.compose.material3.Typography  
import androidx.compose.ui.text.TextStyle  
import androidx.compose.ui.text.font.Font  
import androidx.compose.ui.text.font.FontFamily  
import androidx.compose.ui.text.font.FontWeight  
import androidx.compose.ui.unit.sp  
import es.nuskysoftware.cajamercadillos.R  
  
  
*/\*\*  
 \* Archivo: Type.kt  
 \* Configuramos la tipografía personalizada (Montserrat) para toda la app.  
 \*/*// ------------------- 1. Definimos la familia tipográfica -------------------  
val *Montserrat* = *FontFamily*(  
 *Font*(R.font.*montserrat\_regular*, FontWeight.Normal),  
 *Font*(R.font.*montserrat\_medium*, FontWeight.Medium),  
 *Font*(R.font.*montserrat\_bold*, FontWeight.Bold)  
)  
  
// ------------------- 2. Definimos los estilos tipográficos -------------------  
val *Typography* = Typography(  
 // Título grande (pantallas principales)  
 titleLarge = TextStyle(  
 fontFamily = *Montserrat*,  
 fontWeight = FontWeight.Bold,  
 fontSize = 22.*sp* ),  
  
 // Título medio  
 titleMedium = TextStyle(  
 fontFamily = *Montserrat*,  
 fontWeight = FontWeight.Medium,  
 fontSize = 18.*sp* ),  
  
 // Texto normal  
 bodyLarge = TextStyle(  
 fontFamily = *Montserrat*,  
 fontWeight = FontWeight.Normal,  
 fontSize = 16.*sp* ),  
  
 // Texto pequeño  
 bodyMedium = TextStyle(  
 fontFamily = *Montserrat*,  
 fontWeight = FontWeight.Normal,  
 fontSize = 14.*sp* )  
)  
  
*/\*\*  
 \* Nota:  
 \* - Ahora toda la app usará Montserrat.  
 \* - Solo falta añadir los .ttf en res/font.  
 \*/*

package es.nuskysoftware.cajamercadillos.ui.viewmodel  
  
import androidx.lifecycle.ViewModel  
import androidx.lifecycle.*viewModelScope*import es.nuskysoftware.cajamercadillos.data.local.entity.AvisoUsuarioEntity  
import es.nuskysoftware.cajamercadillos.data.local.entity.AvisoConDetalle  
import es.nuskysoftware.cajamercadillos.data.repository.AvisoRepository  
import kotlinx.coroutines.flow.\*  
import kotlinx.coroutines.launch  
import java.util.UUID  
  
class AvisoViewModel(  
 private val repository: AvisoRepository  
) : ViewModel() {  
  
 val avisosConDetalle: StateFlow<List<AvisoConDetalle>> =  
 repository.getAvisosConDetalle()  
 .*stateIn*(*viewModelScope*, SharingStarted.WhileSubscribed(5000), *emptyList*())  
  
 val avisosNoLeidosCount: StateFlow<Int> =  
 repository.getAvisosNoLeidosCount()  
 .*stateIn*(*viewModelScope*, SharingStarted.WhileSubscribed(5000), 0)  
  
 fun insertarAvisoUsuario(idAvisoBase: Int, mercadilloId: String? = null) {  
 *viewModelScope*.*launch* **{** val nuevoAviso = AvisoUsuarioEntity(  
 idAvisoUsuario = UUID.randomUUID().toString(),  
 idAvisoBase = idAvisoBase,  
 fechaCreacion = System.currentTimeMillis(),  
 leido = false,  
 mercadilloId = mercadilloId  
 )  
 repository.insertAvisoUsuario(nuevoAviso)  
 **}** }  
  
 fun marcarAvisoComoLeido(idAviso: String) {  
 *viewModelScope*.*launch* **{** repository.marcarAvisoComoLeido(idAviso)  
 **}** }  
  
 fun eliminarAvisosPorMercadillo(mercadilloId: String) {  
 *viewModelScope*.*launch* **{** repository.eliminarAvisosPorMercadillo(mercadilloId)  
 **}** }  
  
 // ---------------- SINCRONIZACIÓN FIRESTORE ----------------  
  
 fun sincronizarPendientes() {  
 *viewModelScope*.*launch* **{** *println*(">>> [AvisoViewModel] Sincronizando avisos pendientes...")  
 repository.sincronizarPendientes()  
 **}** }  
  
 fun sincronizarDesdeFirestore() {  
 *viewModelScope*.*launch* **{** *println*(">>> [AvisoViewModel] Descargando avisos desde Firestore...")  
 repository.sincronizarDesdeFirestore()  
 **}** }  
}

package es.nuskysoftware.cajamercadillos.ui.viewmodel  
  
import androidx.lifecycle.ViewModel  
import androidx.lifecycle.*viewModelScope*import es.nuskysoftware.cajamercadillos.data.local.entity.CategoriaEntity  
import es.nuskysoftware.cajamercadillos.data.repository.CategoriaRepository  
import kotlinx.coroutines.flow.MutableStateFlow  
import kotlinx.coroutines.flow.StateFlow  
import kotlinx.coroutines.launch  
  
*/\*\*  
 \* ✅ CategoriaViewModel:  
 \* - Gestiona la lógica para Categorías en modo híbrido (Room + Firestore).  
 \* - Actualiza UI en tiempo real con Flow.  
 \* - Maneja sincronización automática cuando hay conexión.  
 \*/*class CategoriaViewModel(  
 private val repository: CategoriaRepository  
) : ViewModel() {  
  
 // ------------------- ESTADOS PRINCIPALES -------------------  
 private val \_categorias = *MutableStateFlow*<List<CategoriaEntity>>(*emptyList*())  
 val categorias: StateFlow<List<CategoriaEntity>> get() = \_categorias  
  
 val isLoading = *MutableStateFlow*(false)  
 val errorMessage = *MutableStateFlow*<String?>(null)  
  
 init {  
 // ✅ Observa la base local (Room) en tiempo real  
 *viewModelScope*.*launch* **{** repository.getAllCategorias().collect **{** lista **->** \_categorias.value = lista  
 **}  
 }** }  
  
 // ------------------- CRUD -------------------  
  
 */\*\*  
 \* ✅ Insertar nueva categoría (Room + Firestore)  
 \*/* fun insertarCategoria(nombre: String, colorHex: String, orden: Int = 0) {  
 *viewModelScope*.*launch* **{** try {  
 isLoading.value = true  
 val categoria = CategoriaEntity(nombre = nombre, colorHex = colorHex, orden = orden)  
 repository.insertCategoria(categoria)  
 errorMessage.value = null  
 } catch (e: Exception) {  
 errorMessage.value = "Error al insertar: ${e.message}"  
 } finally {  
 isLoading.value = false  
 }  
 **}** }  
  
 */\*\*  
 \* ✅ Actualizar categoría existente  
 \*/* fun actualizarCategoria(categoria: CategoriaEntity) {  
 *viewModelScope*.*launch* **{** try {  
 isLoading.value = true  
 repository.updateCategoria(categoria)  
 errorMessage.value = null  
 } catch (e: Exception) {  
 errorMessage.value = "Error al actualizar: ${e.message}"  
 } finally {  
 isLoading.value = false  
 }  
 **}** }  
  
 */\*\*  
 \* ✅ Eliminar categoría  
 \*/* fun eliminarCategoria(categoria: CategoriaEntity) {  
 *viewModelScope*.*launch* **{** try {  
 isLoading.value = true  
 repository.deleteCategoria(categoria)  
 errorMessage.value = null  
 } catch (e: Exception) {  
 errorMessage.value = "Error al eliminar: ${e.message}"  
 } finally {  
 isLoading.value = false  
 }  
 **}** }  
  
 // ------------------- SINCRONIZACIÓN -------------------  
  
 */\*\*  
 \* ✅ Sincronizar pendientes (Room → Firestore)  
 \*/* fun sincronizar() {  
 *viewModelScope*.*launch* **{** try {  
 isLoading.value = true  
 *println*(">>> [DEBUG] Sincronizando pendientes hacia Firestore...")  
 repository.sincronizarPendientes()  
 errorMessage.value = null  
 } catch (e: Exception) {  
 errorMessage.value = "Error al sincronizar: ${e.message}"  
 } finally {  
 isLoading.value = false  
 }  
 **}** }  
  
 */\*\*  
 \* ✅ Descargar Firestore → Room  
 \*/* fun sincronizarDesdeFirestore() {  
 *viewModelScope*.*launch* **{** try {  
 isLoading.value = true  
 *println*(">>> [DEBUG] Descargando categorías desde Firestore...")  
 repository.sincronizarDesdeFirestore()  
 errorMessage.value = null  
 } catch (e: Exception) {  
 errorMessage.value = "Error al sincronizar desde Firestore: ${e.message}"  
 } finally {  
 isLoading.value = false  
 }  
 **}** }  
}

package es.nuskysoftware.cajamercadillos.ui.viewmodel  
  
import androidx.lifecycle.ViewModel  
import androidx.lifecycle.*viewModelScope*import es.nuskysoftware.cajamercadillos.data.local.entity.ConfiguracionEntity  
import es.nuskysoftware.cajamercadillos.data.repository.ConfiguracionRepository  
import kotlinx.coroutines.flow.MutableStateFlow  
import kotlinx.coroutines.flow.StateFlow  
import kotlinx.coroutines.launch  
  
class ConfiguracionViewModel(  
 private val repository: ConfiguracionRepository  
) : ViewModel() {  
  
 private val \_configuracion = *MutableStateFlow*<ConfiguracionEntity?>(null)  
 val configuracion: StateFlow<ConfiguracionEntity?> get() = \_configuracion  
  
 init {  
 *viewModelScope*.*launch* **{** repository.getConfiguracion().collect **{** \_configuracion.value = **it }  
 }** }  
  
 fun guardarConfiguracion(config: ConfiguracionEntity) {  
 *viewModelScope*.*launch* **{** repository.guardarConfiguracion(config)  
 **}** }  
  
 fun marcarConfigurada() {  
 \_configuracion.value?.*let* **{** actual **->** val actualizado = actual.copy(isConfigurated = true)  
 guardarConfiguracion(actualizado)  
 **}** }  
  
 fun sincronizarDesdeFirestore() {  
 *viewModelScope*.*launch* **{** repository.sincronizarDesdeFirestore()  
 **}** }  
}

package es.nuskysoftware.cajamercadillos.ui.viewmodel  
  
import androidx.lifecycle.ViewModel  
import androidx.lifecycle.ViewModelProvider  
import es.nuskysoftware.cajamercadillos.data.repository.ConfiguracionRepository  
  
*/\*\*  
 \* Factory para crear instancias de ConfiguracionViewModel  
 \* con inyección del repositorio.  
 \*/*class ConfiguracionViewModelFactory(  
 private val repository: ConfiguracionRepository  
) : ViewModelProvider.Factory {  
  
 override fun <T : ViewModel> create(modelClass: Class<T>): T {  
 if (modelClass.isAssignableFrom(ConfiguracionViewModel::class.*java*)) {  
 @Suppress("UNCHECKED\_CAST")  
 return ConfiguracionViewModel(repository) as T  
 }  
 throw IllegalArgumentException("Unknown ViewModel class")  
 }  
}

package es.nuskysoftware.cajamercadillos.utils  
  
  
data class Aviso(val mensaje: String, val tipo: TipoAviso)  
  
enum class TipoAviso {  
 *MERCADILLO\_ABIERTO*,  
 *ARQUEO\_PENDIENTE*}  
  
fun obtenerAvisosSimulados(): List<Aviso> {  
 return *listOf*(  
 Aviso("Tienes un mercadillo abierto sin cerrar.", TipoAviso.*MERCADILLO\_ABIERTO*),  
 Aviso("Hay un arqueo pendiente del último mercadillo.", TipoAviso.*ARQUEO\_PENDIENTE*)  
 )  
}

package es.nuskysoftware.cajamercadillos.utils  
  
import androidx.compose.ui.graphics.Color  
import kotlin.random.Random  
  
fun generarColorAleatorioPastel(): Color {  
 val r = Random.nextInt(150, 256)  
 val g = Random.nextInt(150, 256)  
 val b = Random.nextInt(150, 256)  
 return *Color*(r, g, b)  
}

package es.nuskysoftware.cajamercadillos.utils  
  
import androidx.compose.foundation.layout.fillMaxWidth  
import androidx.compose.foundation.layout.padding  
import androidx.compose.material3.MaterialTheme  
import androidx.compose.material3.Text  
import androidx.compose.runtime.Composable  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.text.style.TextAlign  
import androidx.compose.ui.unit.dp  
import androidx.compose.ui.unit.sp  
  
@Composable  
fun FooterMarca() {  
 Text(  
 text = "® La Nusky Software",  
 fontSize = 14.*sp*,  
 color = MaterialTheme.colorScheme.onBackground.copy(alpha = 0.6f),  
 textAlign = TextAlign.Center,  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*padding*(bottom = 8.*dp*)  
 )  
}

package es.nuskysoftware.cajamercadillos.utils  
  
import android.content.Context  
import android.net.ConnectivityManager  
import android.net.Network  
import android.net.NetworkCapabilities  
import kotlinx.coroutines.flow.MutableStateFlow  
import kotlinx.coroutines.flow.StateFlow  
  
*/\*\*  
 \* ✅ NetworkMonitor con StateFlow:  
 \* Detecta cambios de red en tiempo real y expone un flujo reactivo.  
 \*/*class NetworkMonitor(context: Context) {  
  
 private val \_isConnected = *MutableStateFlow*(false)  
 val isConnected: StateFlow<Boolean> get() = \_isConnected  
  
 private val connectivityManager =  
 context.getSystemService(Context.*CONNECTIVITY\_SERVICE*) as ConnectivityManager  
  
 private val networkCallback = object : ConnectivityManager.NetworkCallback() {  
 override fun onAvailable(network: Network) {  
 \_isConnected.value = true  
 }  
  
 override fun onLost(network: Network) {  
 \_isConnected.value = false  
 }  
 }  
  
 init {  
 connectivityManager.registerDefaultNetworkCallback(networkCallback)  
  
 val activeNetwork = connectivityManager.*activeNetwork* val capabilities = connectivityManager.getNetworkCapabilities(activeNetwork)  
 \_isConnected.value =  
 capabilities?.hasCapability(NetworkCapabilities.*NET\_CAPABILITY\_INTERNET*) == true  
 }  
}

package es.nuskysoftware.cajamercadillos.work  
  
import android.content.Context  
import androidx.work.CoroutineWorker  
import androidx.work.WorkerParameters  
import com.google.firebase.firestore.FirebaseFirestore  
import es.nuskysoftware.cajamercadillos.data.local.database.AppDatabase  
import kotlinx.coroutines.tasks.await  
  
*/\*\*  
 \* CategoriaSyncWorker:  
 \* ✅ Encargado de sincronizar categorías pendientes con Firestore.  
 \* ✅ Corre en segundo plano (WorkManager) y se reintenta si no hay conexión.  
 \*  
 \* Flujo:  
 \* 1. Consultar Room → obtener categorías con pendienteSincronizar = true.  
 \* 2. Intentar subir cada una a Firestore.  
 \* 3. Si sube bien → marcar como sincronizada en Room.  
 \*/*class CategoriaSyncWorker(  
 context: Context,  
 params: WorkerParameters  
) : CoroutineWorker(context, params) { // CoroutineWorker = permite usar corrutinas  
  
 override suspend fun doWork(): Result {  
 try {  
 // 1️⃣ Instanciamos la base de datos local (Room)  
 val db = AppDatabase.getDatabase(*applicationContext*)  
 val categoriaDao = db.categoriaDao()  
  
 // 2️⃣ Instanciamos Firestore  
 val firestore = FirebaseFirestore.getInstance()  
  
 // 3️⃣ Obtenemos todas las categorías pendientes de sincronizar  
 val pendientes = categoriaDao.getCategoriasPendientes()  
  
 // 4️⃣ Recorremos y subimos cada categoría  
 for (categoria in pendientes) {  
 try {  
 firestore.collection("categorias")  
 .document(categoria.idCategoria)  
 .set(categoria)  
 .await() // ✅ Espera a que la subida termine  
  
 // ✅ Si subió OK → marcar como sincronizada en Room  
 categoriaDao.marcarComoSincronizada(categoria.idCategoria)  
 } catch (e: Exception) {  
 // Si falla UNA, seguimos con las demás  
 }  
 }  
  
 // ✅ *Todo bien → devolvemos SUCCESS* return Result.success()  
  
 } catch (e: Exception) {  
 // Si algo grande falla → reintentar después  
 return Result.retry()  
 }  
 }  
}

package es.nuskysoftware.cajamercadillos  
  
import android.os.Bundle  
import androidx.activity.ComponentActivity  
import androidx.activity.compose.setContent  
import androidx.compose.runtime.\*  
import androidx.lifecycle.ViewModel  
import androidx.lifecycle.ViewModelProvider  
import androidx.lifecycle.viewmodel.compose.viewModel  
import androidx.navigation.NavHostController  
import androidx.navigation.compose.NavHost  
import androidx.navigation.compose.composable  
import androidx.navigation.compose.rememberNavController  
import androidx.room.Room  
import com.google.firebase.FirebaseApp  
import com.google.firebase.firestore.FirebaseFirestore  
import es.nuskysoftware.cajamercadillos.data.local.database.AppDatabase  
import es.nuskysoftware.cajamercadillos.data.repository.AvisoRepository  
import es.nuskysoftware.cajamercadillos.data.repository.CategoriaRepository  
import es.nuskysoftware.cajamercadillos.data.repository.ConfiguracionRepository  
import es.nuskysoftware.cajamercadillos.ui.pantallas.\*  
import es.nuskysoftware.cajamercadillos.ui.theme.CajaMercadillosTheme  
import es.nuskysoftware.cajamercadillos.ui.viewmodel.\*  
import es.nuskysoftware.cajamercadillos.utils.NetworkMonitor  
import androidx.compose.runtime.collectAsState  
  
class MainActivity : ComponentActivity() {  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
  
 // ✅ Inicializar Firebase  
 FirebaseApp.initializeApp(this)  
  
 // ✅ Instancia Room (BD local)  
 val db = Room.databaseBuilder(  
 *applicationContext*,  
 AppDatabase::class.*java*,  
 "caja\_mercadillos\_db"  
 ).fallbackToDestructiveMigration().build()  
  
 // ✅ Instancia Firestore (BD nube)  
 val firestore = FirebaseFirestore.getInstance()  
  
 // ✅ Repositorios  
 val categoriaRepository = CategoriaRepository(db.categoriaDao(), firestore)  
 val configuracionRepository = ConfiguracionRepository(db.configuracionDao(), firestore)  
 val avisoRepository = AvisoRepository(  
 avisoBaseDao = db.avisoBaseDao(),  
 avisoUsuarioDao = db.avisoUsuarioDao(),  
 firestore = firestore  
 )  
  
 // ✅ Monitor de red  
 val networkMonitor = NetworkMonitor(*applicationContext*)  
  
 *setContent* **{** CajaMercadillosTheme **{** val navController = rememberNavController()  
  
 // ✅ ViewModel Categorías  
 val categoriaViewModel: CategoriaViewModel = viewModel(  
 factory = CategoriaViewModelFactory(categoriaRepository)  
 )  
  
 // ✅ ViewModel Configuración  
 val configuracionViewModel: ConfiguracionViewModel = viewModel(  
 factory = ConfiguracionViewModelFactory(configuracionRepository)  
 )  
  
 // ✅ ViewModel Avisos  
 val avisoViewModel: AvisoViewModel = viewModel(  
 factory = AvisoViewModelFactory(avisoRepository)  
 )  
  
 // ✅ Estado de conexión  
 val isConnected by networkMonitor.isConnected.collectAsState(initial = false)  
  
 // ✅ Sincronización automática cuando hay conexión  
 LaunchedEffect(isConnected) **{** if (isConnected) {  
 *println*(">>> [DEBUG] Conexión activa: sincronizando datos...")  
  
 // Categorías  
 categoriaViewModel.sincronizar()  
 categoriaViewModel.sincronizarDesdeFirestore()  
  
 // Configuración  
 configuracionViewModel.sincronizarDesdeFirestore()  
  
 // Avisos  
 avisoViewModel.sincronizarPendientes()  
 avisoViewModel.sincronizarDesdeFirestore()  
  
 *println*(">>> [OK] Sincronización completa (categorías + configuración + avisos).")  
 } else {  
 *println*(">>> [DEBUG] Sin conexión, trabajando en modo offline...")  
 }  
 **}** // ✅ Navegación principal  
 AppNavigation(  
 navController = navController,  
 categoriaViewModel = categoriaViewModel,  
 configuracionViewModel = configuracionViewModel,  
 avisoViewModel = avisoViewModel  
 )  
 **}  
 }** }  
}  
  
// ✅ Factory para ViewModel de Categorías  
class CategoriaViewModelFactory(  
 private val repository: CategoriaRepository  
) : ViewModelProvider.Factory {  
 override fun <T : ViewModel> create(modelClass: Class<T>): T {  
 if (modelClass.isAssignableFrom(CategoriaViewModel::class.*java*)) {  
 @Suppress("UNCHECKED\_CAST")  
 return CategoriaViewModel(repository) as T  
 }  
 throw IllegalArgumentException("Unknown ViewModel class")  
 }  
}  
  
// ✅ Factory para ViewModel de Avisos  
class AvisoViewModelFactory(  
 private val repository: AvisoRepository  
) : ViewModelProvider.Factory {  
 override fun <T : ViewModel> create(modelClass: Class<T>): T {  
 if (modelClass.isAssignableFrom(AvisoViewModel::class.*java*)) {  
 @Suppress("UNCHECKED\_CAST")  
 return AvisoViewModel(repository) as T  
 }  
 throw IllegalArgumentException("Unknown ViewModel class")  
 }  
}  
  
// ✅ Navegación principal  
@Composable  
fun AppNavigation(  
 navController: NavHostController,  
 categoriaViewModel: CategoriaViewModel,  
 configuracionViewModel: ConfiguracionViewModel,  
 avisoViewModel: AvisoViewModel  
) {  
 NavHost(navController = navController, startDestination = "splash") **{** *composable*("splash") **{** PantallaSplash(  
 navController = navController,  
 configuracionViewModel = configuracionViewModel  
 )  
 **}** *composable*("home") **{** PantallaHome(navController, avisoViewModel)  
 **}** *composable*("articulos\_categorias") **{** PantallaArticulosCategorias(navController, avisoViewModel)  
 **}** *composable*("categorias") **{** PantallaCategorias(navController, categoriaViewModel, avisoViewModel)  
 **}** *composable*("pantalla\_avisos") **{** PantallaAvisos(  
 navController = navController,  
 viewModel = avisoViewModel  
 )  
 **}** *composable*("configuracion") **{** PantallaConfiguracion(  
 navController = navController,  
 viewModel = configuracionViewModel  
 )  
 **}  
 }**}

Texto

El contenido generado por IA puede ser incorrecto.