DULCERIA

Nombre del alumno: José Manuel Monsa Jimeno
Ciclo Formativo: Desarrollo de Aplicaciones Multiplataforma (DAM)
Curso y convocatoria: 2024-2025 / Convocatoria Junio
Centro: IES San José (Cortegana)
Tutor : Azcárate Llanes, Francisco de Asís

Contenido

2. RESUMEN

Este documento presenta una aplicación para gestionar una tienda de golosinas ("Dulcería"). La solución consta de un pequeño servidor que almacena la información (backend en PHP + MySQL) y una app en Table en Android. El sistema permite:

- Registrar y mantener el catálogo de productos (nombre, tamaño, precio, marca).
- Gestionar el inventario en almacén con fechas de caducidad individualizadas.
- Visualizar y filtrar productos por marca y tamaño, así como consultar el stock agrupado.
- Agregar, editar y eliminar productos, así como consumir unidades de producto (restar stock).
- Programar notificaciones dos semanas antes de la fecha de caducidad de cada lote.
- Ofrecer una interfaz de usuario intuitiva basada en Material Design, con interacciones típicas de RecyclerView y diálogos emergentes.

El objetivo principal ha sido diseñar una solución sencilla de desplegar en entornos Android. Se ha utilizado la elección de PHP y MySQL para el backend por su bajo coste de despliegue y facilidad de alojamiento, junto con Kotlin y Retrofit para el cliente, asegurando un mantenimiento ágil y una futura ampliación.

---

3. PALABRAS CLAVE

- Gestión de Inventario
- Backend PHP
- MySQL
- Kotlin
- Android
- Retrofit
- WorkManager
- RecyclerView
- Notificaciones Push

---

4. INTRODUCCIÓN

4.1. Descripción breve del proyecto y su objetivo

En la actualidad, las tiendas de golosinas requieren herramientas digitales ligeras que permitan controlar en tiempo real su catálogo de productos y el estado del inventario, especialmente cuando cada lote de producto tiene fecha de caducidad. El proyecto "Dulcería" propone desarrollar una plataforma sencilla de gestión compuesta por un backend web (PHP + MySQL) y una aplicación en Android (Kotlin), que facilite a los responsables de la tienda:

1. Mantener el catálogo de productos (nombre, tamaño, precio, marca).
2. Registrar y consultar fechas de caducidad individuales de cada unidad (almacén).
3. Visualizar alertas automáticas (notificaciones) cuando un lote esté próximo a caducar.
4. Automatizar el flujo de inserción, edición y eliminación de productos desde la app.

4.2. Problemática actual y soluciones existentes

Muchas tiendas tradicionales siguen gestionando el inventario en hojas de cálculo o mediante anotaciones manuales, lo cual provoca errores humanos y falta de alertas tempranas sobre caducidades. Las soluciones comerciales suelen ser costosas o demasiado complejas para un negocio pequeño. Este proyecto busca una alternativa gratuita, de código abierto y fácil de desplegar en un servidor local o alojamiento compartido, integrando de forma nativa la experiencia de usuario en Android.

4.3. Objetivo principal y propuesta de mejora

El proyecto persigue la creación de una herramienta que:

- Centralice la gestión de productos y almacén en un único sistema.
- Permita automatizar la creación de alertas de caducidad sin intervención del usuario.
- Simplifique la operativa diaria del responsable de la tienda mediante una interfaz clara.

4.4. Planteamiento de soluciones y justificación

Se evaluaron distintas tecnologías para el backend como solo usar MySQL o SQLite, optándose finalmente por **PHP + MySQL** debido a:

- Facilidad de configuración en la mayoría de alojamientos compartidos.
- Gran cantidad de documentación y recursos.
- Bajo consumo de recursos para las consultas básicas que requiere esta aplicación.
- Más la gran robustez de MySQL .

Se eligió **Kotlin** porque:

- Aprovecha el ecosistema nativo de Android (mejor rendimiento, acceso directo a APIs).
- RetroAlimenta más fácilmente con librerías como Retrofit o WorkManager.

---

5. ESTADO DEL ARTE

5.1. Trabajos y herramientas relacionadas

- **Sistema TPV open-source**: Existen proyectos de punto de venta (TPV) gratuitos basados en PHP (por ejemplo, uniCenta o OpenCart), pero su complejidad supera la de una tienda pequeña como la que he realizado en este caso.
- **Apps móviles de gestión de stock**: Algunas apps en Google Play enfocadas a inventarios genéricos, pero con suscripciones de pago y sin personalización de caducidad por lote.

5.2. Tecnologías y lenguajes analizados

- **Backend**:
  - Node.js + Express + MongoDB: ligero, pero requiere un entorno distinto (servidor Node), normalmente más costoso en hosting compartido.
  - Python + Django + PostgreSQL: robusto, pero mayor complejidad de configuración en servidores convencionales.
  - PHP + MySQL: elegido por ser ofrecido en la mayoría de hosting, con un despliegue trivial (archivos PHP + base de datos MySQL).
- **Frontend móvil**:
  - Flutter + Dart: permite Android+iOS; sin embargo, los conocimientos previos estan más orientados a Java/Kotlin.
  - React Native: similar a Flutter pero con dependencia de Node.js y npm; se descartó por preferir acceder a APIs nativas de Android (WorkManager, notificaciones) fácilmente.
  - Incorporar escáner de códigos de barras: Para acelerar la introducción de productos, se plantea usar un lector de código de barras que rellene automáticamente los datos, haciendo el alta más ágil y sencilla.
  - Kotlin nativo: se aprovechó Android Studio, Material Design y librerías comunes (Retrofit, Gson, WorkManager), obteniendo un APK ligero y sin dependencias externas.

---

6. ESTUDIO DE VIABILIDAD

6.1. DAFO

Fortalezas	Debilidades
• Código abierto y gratuito, sin licencias comerciales.	• Funcionalidades limitadas a tiendas pequeños; no cubre TPV ni gestión de ventas.
• Despliegue sencillo en LAMP (la mayoría de hostings lo soportan).	• Ausencia de versión iOS; solo Android.

Fortalezas	Debilidades
• Notificaciones locales basadas en WorkManager; no requiere servidor adicional.	• Diseño básico de interfaz; no contempla temas de accesibilidad avanzados.
• Estructura de base de datos flexible para adaptar futuras ampliaciones.	• Seguridad mínima (sin autenticación de usuarios ni roles).
Oportunidades	**Amenazas**
• Implementar en pequeñas tiendas que carezcan de presupuesto para TPV.	• Competencia de soluciones comerciales con funciones extra (facturación, pagos).
• Posibilidad de añadir módulos de facturación y pasarela de pago en el futuro.	• Cambios en políticas de permisos de Android (notificaciones) que requieran ajustes.
• Creciente adopción de dispositivos Android en entornos de retail.	• Obsolescencia de tecnologías (por ejemplo, PHP 7.x frente a PHP 8.x).

6.2. Estudio de mercado

Esta aplicación está diseñada para tiendas pequeñas que no dispongan de recursos suficientes para adquirir un TPV comercial. Con nuestra solución, no se requieren infraestructuras costosas: basta con un servidor básico que soporte PHP y MySQL. Ofrece funcionalidades optimizadas para la operativa diaria de una tienda, garantizando robustez y simplicidad (archivos PHP + base de datos MySQL).

6.2.1. Viabilidad temporal

La duración estimada de este proyecto (análisis, diseño, desarrollo y pruebas) es de 4 meses, distribuidos aproximadamente de esta forma:

- **Análisis de requisitos y diseño**: 2 semana
- **Desarrollo backend**: 1-2 semana
- **Desarrollo de la app Android**: 2 meses
- **Pruebas, ajustes y documentación**: 1-2 semana

---

7. ANÁLISIS DE REQUISITOS

7.1. Requisitos funcionales

1. El sistema debe permitir al administrador:
    - Ver listado de productos existentes (filtrables por marca y tamaño).
    - Insertar nuevos productos.
    - Editar datos de un producto (nombre, tamaño, precio, marca, estado “disponible”).
    - Eliminar productos (marcar no disponibles).
2. El sistema debe gestionar el inventario en almacén por fechas de caducidad:
    - Añadir fechas de caducidad por unidad al insertar un producto en almacén.

   - Mostrar en pantalla el listado de inventario agrupado por producto (cantidad y próxima fecha de caducidad).
   - Permitir eliminar una unidad específica de inventario ("consumir producto").
3. El sistema debe generar notificaciones automáticas en Android:
   - Programar alertas dos semanas antes de que una unidad alcance su fecha de caducidad.
   - Mostrar notificaciones locales con título y mensaje del producto caducará pronto.
4. La app Android debe conectarse al backend a través de APIs REST:
   - Obtener productos disponibles (getProductos.php).
   - Obtener datos de inventario (getAlmacen.php).
   - Insertar producto (addProducto.php).
   - Insertar fechas de almacén en producto existente (addAlmacen.php).
   - Editar producto y su almacén (editProducto.php).
   - Eliminar producto y sus lotes (deleteProducto.php).
   - Cambiar estado "disponible" de producto (available.php).
   - Consumir unidad de inventario (consumeProduct.php).
5. La app en Table debe ofrece:
   - Interfaz de selección de marca (botones con iconos).
   - Filtrado por tamaño (pequeño / grande).
   - Visualización de detalle del producto y resumen de pedido (cantidad y total).
   - Acceso a pantalla de gestión de almacén (StoreActivity) con filtros por nombre, marca, tamaño y fecha límite, mostrando en forma de tabla.
   - Formulario de "Añadir Producto" con validaciones y generación dinámica de selectores de fecha.
   - Diálogo emergente para editar o eliminar en la vista de almacén.

7.2. Requisitos no funcionales

1. **Usabilidad:** La interfaz debe ser intuitiva y responsiva, con animaciones mínimas y botones claramente etiquetados.
2. **Rendimiento:**
   - El tiempo de respuesta de la API PHP no debe superar 500 ms en operaciones de lectura/escritura básicas con una base de datos .
   - La carga de pantallas en la app Android debe ser inferior a 1 segundo, salvo la primera descarga de datos que puede llegar a 2 segundos.
3. **Seguridad:**
   - Las consultas SQL deben emplear sentencias preparadas para prevenir inyecciones SQL.
   - El backend usará Content-Type: application/json con UTF-8.
   - No se implementa autenticación de usuarios en esta versión (futura mejora).
4. **Mantenibilidad:**
   - El código PHP y Kotlin debe estar documentado (comentarios y Javadoc donde corresponda).
   - La estructura de carpetas debe separar claramente Model, Api, RecyclerView, Notification y Activity.
5. **Portabilidad:**

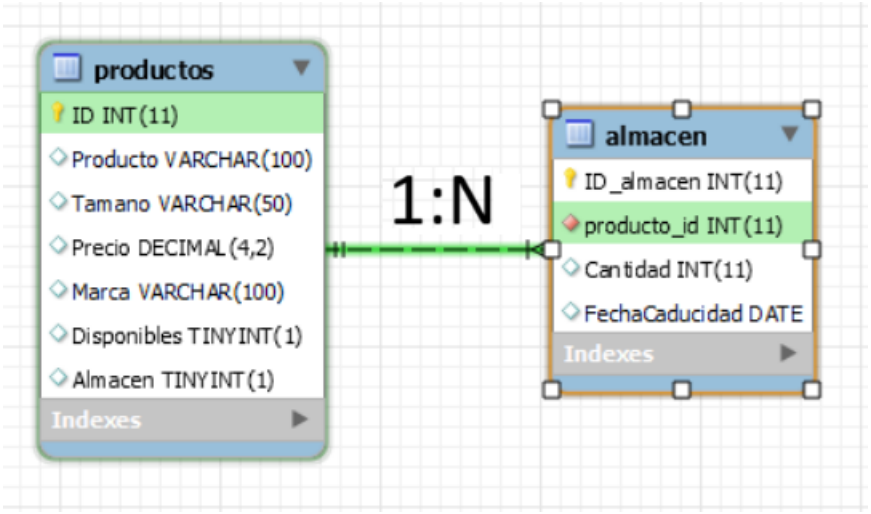
- El backend se despliega en un servidor LAMP (Linux, Apache, MySQL, PHP ≥ 7.4).
- La app Android es compatible con API nivel 21 (Android 5.0) en adelante.

6. **Compatibilidad:**
   - Las librerías usadas (Retrofit 2.9.0, Gson 2.8.x, WorkManager de AndroidX) deben ser compatibles con la versión mínima de Android (API 21).

---

8. DISEÑO

8.1. Diseño Conceptual Entidad-Relación (DER)



- **Entidad productos**: contiene atributos básicos de cada producto.
- **Entidad almacen**: cada fila representa una unidad de producto con su fecha de caducidad respectiva.

8.2. Diseño Lógico Relacional

Tras pasar de un modelo conceptual ER a tablas relacionales, se obtienen:

1. **Tabla productos**
   - ID INT AUTO_INCREMENT PRIMARY KEY
   - Producto VARCHAR(100) NOT NULL
   - Tamano VARCHAR(50) NOT NULL
   - Precio DECIMAL(4,2) NOT NULL
   - Marca VARCHAR(100) NOT NULL
   - Disponibles BOOLEAN NOT NULL DEFAULT 1
   - Almacen BOOLEAN NOT NULL DEFAULT 0
2. **Tabla almacen**

- ID_almacen INT AUTO_INCREMENT PRIMARY KEY
- producto_id INT NOT NULL, FOREIGN KEY REFERENCES productos(ID) ON DELETE CASCADE
- Cantidad INT NOT NULL
- FechaCaducidad DATE NOT NULL

8.3. Diseño Físico (Esquema MySQL y optimizaciones)

8.3.1. Esquema de creación de la base de datos

```
CREATE DATABASE IF NOT EXISTS tienda;
USE tienda;

CREATE TABLE productos (
  ID INT NOT NULL AUTO_INCREMENT,
  Producto    VARCHAR(100),
  Tamano      VARCHAR(50),
  Precio      DECIMAL(4,2),
  Marca       VARCHAR(100),
  Disponibles BOOLEAN,
  Almacen     BOOLEAN,
  PRIMARY KEY (ID)
);

CREATE TABLE almacen (
  ID_almacen     INT NOT NULL AUTO_INCREMENT,
  producto_id    INT NOT NULL,
  Cantidad       INT,
  FechaCaducidad DATE,
  PRIMARY KEY (ID_almacen),
  FOREIGN KEY (producto_id) REFERENCES productos(ID)
);
```

- **Índices**:
  - Por defecto, MySQL crea un índice sobre ID de ambas tablas.
  - Para optimizar consultas frecuentes (por ejemplo, consultas por marca o fecha de caducidad), se podrían crear índices adicionales
- **Optimización**: Se eligió para garantizar integridad referencial y transacciones (p. ej., en editProducto.php).

8.4. Descripción de tablas y campos

- **productos**
  - ID: Clave primaria.
  - Producto: Nombre del producto (p. ej., “Chocolate con Leche”).
  - Tamano: Tamaño del envase (“Pequeño” / “Grande”).

  - Precio: Precio unitario, en euros, con dos decimales.
  - Marca: Marca comercial (p. ej., "Churruca", "Fini").
  - Disponibles: Bandera (0/1) que indica si el producto está activo/visible en el catálogo.
  - Almacen: Bandera (0/1) que indica si el producto actualmente tiene stock en almacén.
- **almacen**
  - ID_almacen: Clave primaria de cada registro.
  - producto_id: Clave foránea hacia productos.ID, indica a qué producto pertenece este lote.
  - Cantidad: Siempre se inserta como 1 por registro (se almacena una fila por cada unidad).
  - FechaCaducidad: Fecha de caducidad de esa unidad, formateada en DD/MM/AAAA en las APIs.

8.5. Orientación a Objetos

8.5.1. Diagramas de clases (Kotlin y PHP)

1. **Clases Kotlin**
   - **RetrofitClient** (single-ton)
     - instance: ApiService
   - **ApiService** (interfaz)
     - getProductosPorMarca(marca: String): Call<List<Producto>>
     - getAlmacen(): Call<List<Almacen>>
     - addProducto(req: AddProductoRequest): Call<AddProductoResponse>
     - addAlmacen(req: AddAlmacenRequest): Call<AddAlmacenResponse>
     - deleteProducto(id: Int): Call<AddProductoResponse>
     - available(id: Int, disponibles: Int): Call<AddProductoResponse>
     - editProducto(...)
     - consumeProduct(id: Int): Call<ConsumeResponse>
   - **Producto** (data class)
     - id: Int
     - producto: String
     - tamano: String
     - precio: Double
     - marca: String
     - disponibles: Int
     - almacen: Int
   - **Almacen** (data class)
     - id: Int
     - producto: String
     - tamano: String
     - precio: Double
     - marca: String
     - disponibles: Int
     - almacen: Int
     - cantidad: Int?
     - fechaCaducidad: String?
   - **MainActivity**
     - Propiedades: binding: ActivityMainBinding, adapter: ProductosAdapter, currentProductos: List<Producto>, currentSizeFilter: String, resumen: MutableMap<String, Pair<Double, Int>>

        - Métodos: onCreate(), cargar(marca: String?), aplicarFiltro(), mostrarDetalle(p: Producto), onProductoClicked(p: Producto), renderizarResumen(), limpiarResumen()
    - **AddProductActivity**
        - Propiedades: binding: ActivityAddProductBinding, tamaños: List<String>, marcas: List<String>, fechasList: MutableList<String>, sdf: SimpleDateFormat
        - Métodos: onCreate(), generateDatePickers(count: Int), showDatePicker(index: Int, tv: TextView), checkOrCreateProducto(), showConfirmAddAlmacen(productoId: Int, cantidad: Int, fechasCsv: String), addOnlyAlmacen(productoId: Int, cantidad: Int, fechasCsv: String)
    - **StoreActivity**
        - Propiedades: binding: ActivityStoreBinding, allItems: List<Almacen>, displaySdf: SimpleDateFormat
        - Data Class interna: GroupItem(id: Int, product: String, brand: String, size: String, count: Int, nextDate: String?)
        - Métodos: onCreate(), loadAlmacen(), resetFilters(), showBrandMenu(), showSizeMenu(), pickCutoffDate(), applyFilters(), showInTable(items: List<Almacen>)
    - **ProductosAdapter**
        - Propiedades: productos: List<Producto>, context: Context, onItemClick: (Producto) -> Unit
        - Métodos: onCreateViewHolder(), onBindViewHolder(), getItemCount(), updateProductos(newProductos: List<Producto>)
        - **ViewHolder** anidado: vincula ItemProductoBinding, expone bind(p: Producto, onItemClick: (Producto) -> Unit)
    - **AlmacenItemActions**
        - Propiedades: context: Context, tamaños: List<String>, marcas: List<String>
        - Métodos: bindRowClick(rowView: View, item: Almacen, onUpdated: () -> Unit), confirmDelete(item: Almacen, onDeleted: () -> Unit), confirmConsume(item: Almacen, onConsumed: () -> Unit), showEditDialog(item: Almacen, onUpdated: () -> Unit)
    - **NotificationWorker** (Worker de WorkManager)
        - Propiedades: —
        - Métodos: doWork(): Result
        - Companion: programar (context: Context, item: Almacen) (calcula retraso y programa OneTimeWorkRequest)
2. **Clases PHP (backend)**
    - **getProductos.php**
        - Conexión MySQLi, consulta a tabla productos, filtra opcionalmente por marca, muestra JSON de productos con campo Disponibles = 1.
    - **getAlmacen.php**
        - Conexión MySQLi, LEFT JOIN entre productos y almacen para todos los productos con Almacen = 1, devuelve JSON con número de unidades y fecha formateada.
    - **addProducto.php**
        - Lee JSON del cuerpo, valida campos (producto, tamaño, precio, marca, disponibles, almacen).
        - Comprueba si ya existe un producto con mismo nombre/tamaño/marca:
            - Si existe → devuelve { status: "exists", existingId: xx }.

        - Si no existe → inserta en productos y, si almacen=1, inserta tantas filas en almacen como fechas indica.
    - **addAlmacen.php**
        - Lee JSON, valida producto_id, cantidad, fechaCaducidad (CSV).
        - Inserta en almacen una fila por cada fecha proporcionada en la cadena CSV.
    - **editProducto.php**
        - Recibe via POST campos: id, producto, tamano, precio, marca, disponibles, almacen, cantidad, fechaCaducidad.
        - Valida, abre transacción MySQLi, actualiza tabla productos.
        - Si almacen=1: intenta UPDATE almacen SET Cantidad=..., FechaCaducidad=... WHERE producto_id = ?; si no afectó filas → INSERT NEW.
        - Si almacen=0: elimina todas las filas de almacen con producto_id.
        - Commit / rollback según corresponda.
    - **available.php**
        - Recibe POST (id, disponibles), actualiza productos.Disponibles.
    - **deleteProducto.php**
        - Recibe POST (id), abre transacción, primero borra todas las filas de almacen con producto_id, luego elimina el producto de productos.
    - **consumeProduct.php**
        - Recibe POST (id), abre transacción, selecciona la fila de almacen con fecha mínima para ese producto_id, la elimina.
        - Después cuenta cuántas filas quedan y obtiene la nueva fecha mínima.

8.5.2. Descripción de clases y atributos

- **Producto (Kotlin)**
    - id: Identificador único del producto en la base de datos.
    - producto: Nombre textual (p. ej., "Gominolas de Fruta").
    - tamano: Tamaño ("Pequeño" o "Grande").
    - precio: Precio unitario (Double).
    - marca: Marca de la golosina (p. ej., "Fini").
    - disponibles: Indicador (0/1) si está activo en catálogo.
    - almacen: Indicador (0/1) si tiene stock en almacén.
- **Almacen (Kotlin)**
    - id: Identificador único de la unidad en almacén (ID_almacen).
    - producto: Nombre del producto (se reenvía el campo de JOIN con productos).
    - tamano: Tamaño del producto.
    - precio: Precio unitario (para cálculos de total al agrupar).
    - marca: Marca del producto.
    - disponibles: Indicador si aún está activo en catálogo.
    - almacen: Indicador si está en almacén (debería ser siempre 1 para mostrar en StoreActivity).
    - cantidad: Siempre 1 por unidad, pero se reutiliza para indicar que existe alguna unidad.
    - fechaCaducidad: Fecha formateada ("dd/MM/yyyy") de caducidad.

8.6. Diagramas de secuencia (extracto de los procesos más relevantes)

8.6.1. Secuencia: Carga de productos por marca (MainActivity → Backend)

https://lucid.app/lucidspark/d28e364c-3af8-4e8c-bd3b-94dba04c857d/edit?view_items=J7Sltjxx_YmD&invitationId=inv_68173ad0-1387-46ee-a6f6-8b981c4d07fd

8.6.2. Secuencia: Añadir producto

https://lucid.app/lucidspark/d28e364c-3af8-4e8c-bd3b-94dba04c857d/edit?view_items=J7Sltjxx_YmD&invitationId=inv_68173ad0-1387-46ee-a6f6-8b981c4d07fd

8.6.3. Secuencia: Consumir unidad de inventario

https://lucid.app/lucidchart/61de5cf3-a922-4d3e-9a40-2c8e2ac0df83/edit?viewport_loc=-1018%2C-59%2C3340%2C2825%2C0_0&invitationId=inv_2fb041e5-e4b0-4230-93ed-8378daab2d54

8.7. Diseño UX

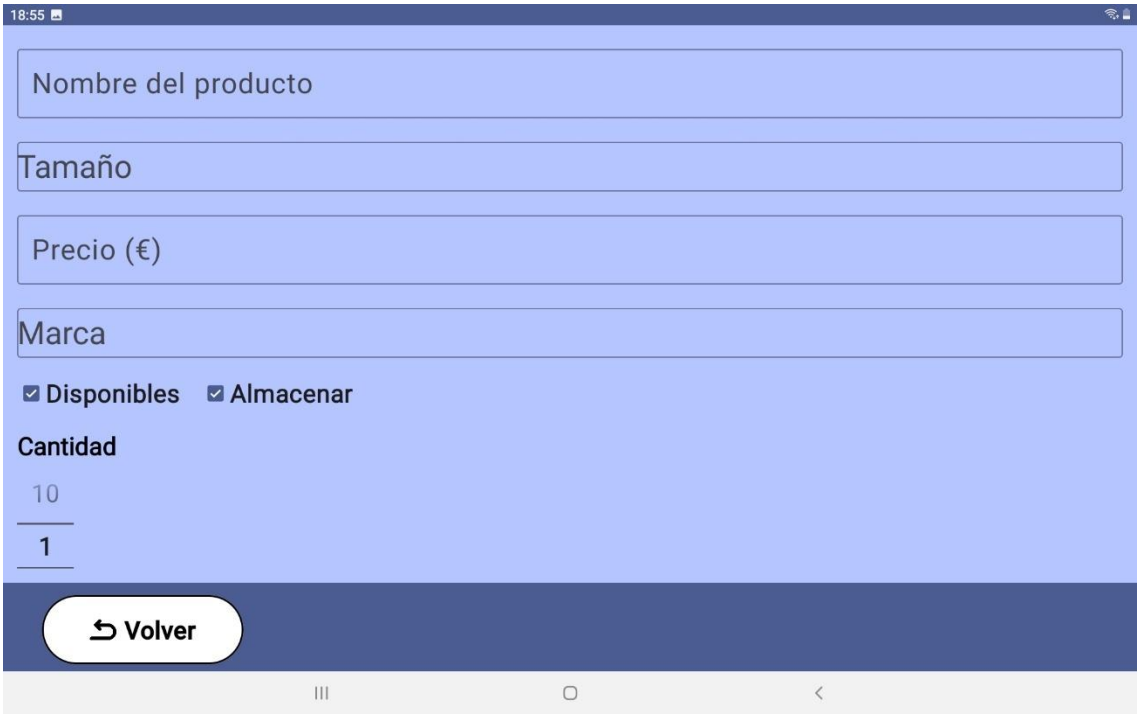
1. **Colores y estilo**:
   - Se utiliza la paleta de Material Design (tonos primarios e inversos) para distinguir botones de marca y fondo de pantallas.
   - El fondo general es un color claro (md_theme_inversePrimary), con elementos destacados en md_theme_primary para controles interactivos.
2. **Navegación**:
   - **MainActivity**: permite seleccionar marca (scroll horizontal), filtrar por tamaño (botones “Pequeño” / “Grande”) y ver resumen de pedido a la derecha.
   - **AddProductActivity**: formulario vertical con campos obligatorios en primer bloque, luego sección de almacén que aparece solo si el usuario marca “Almacenar”.
   - **StoreActivity**: tabla scrollable con filtros en parte superior (nombre, marca, tamaño, fecha), botón “Volver” en la parte inferior.
3. **Componentes**:
   - **RecyclerView con GridLayout (3 columnas)** en MainActivity para mostrar productos tipo “tarjeta” (CardView con nombre, precio y botón de eliminar).
   - **TableLayout** en StoreActivity para mostrar inventario agrupado (cantidad, producto, marca, tamaño, próxima caducidad).
   - **PopupMenu** para acciones contextuales (Modificar, Eliminar, Consumir) al pulsar sobre una fila de inventario.
   - **MaterialCheckBox** para opciones binarias (Disponibles, Almacenar).

- **AutoCompleteTextView** para facilitar introducción de tamaño y marca.

4. **Mockups**
   (En este apartado se añadirán capturas de pantalla de cada pantalla de la app desempeñada en un emulador o dispositivo real. Aquí se dejan referencias a imágenes que se adjuntarán en el anexo.)

- Mockup 1: **Pantalla principal (MainActivity)** con scroll de marcas, grid de productos y resumen.

- Mockup 2: **Formulario "Añadir Producto"** con ejemplos de selección de tamaño, marca, número de fechas y DatePicker.



- Mockup 3: **Vista de almacén (StoreActivity)** mostrando la tabla con filtros aplicados y menú emergente sobre fila.

9. CODIFICACIÓN

9.1. Tecnologías elegidas y justificación

- **Lenguaje Backend**: PHP 7.4+
  - Justificación: Amplio soporte en alojamientos compartidos, despliegue sencillo.
- **Base de Datos**: MySQL
  - Justificación: Sencillo de instalar en XAMPP o hosting compartido. Garantiza integridad referencial con claves foráneas.
- **Lenguaje Frontend Móvil**: Kotlin (Android)
  - Justificación: Permite aprovechar APIs nativas (WorkManager, Retrofit, RecyclerView). Buen rendimiento y mantenimiento a largo plazo.
  - IDE: Android Studio.
- **Librerías y Frameworks**:
  - **Retrofit 2.9.0 + GsonConverter**: para consumo de APIs REST con mapeo automático de JSON a objetos (data classes).
  - **OkHttp Logging-Interceptor**: para depuración de peticiones HTTP en tiempo de desarrollo.
  - **WorkManager (AndroidX)**: para programar notificaciones de caducidad en segundo plano, garantizando ejecución incluso si la app no está abierta.
  - **Material Components (Material3)**: para widgets (TextInputLayout, MaterialButton, MaterialCheckBox) en interfaces responsivas.
  - **RecyclerView + CardView**: para mostrar listados dinámicos de productos.
  - **ConstraintLayout** y **TableLayout** para optimizar uso de la pantalla en distintos tamaños.

9.2. Desarrollo de servicios (Backend)

9.2.1. Descripción general

- Cada script PHP se aloja en la ruta /api en el servidor.
- Utiliza extensiones nativas de PHP (mysqli) sin añadir frameworks de terceros para reducir complejidad.
- Se emplean transacciones en las operaciones de edición y eliminación (scripts editProducto.php, deleteProducto.php) para mantener consistencia entre tablas.

9.2.2. Seguridad

- Todas las entradas de usuario (en consultas GET)
- Los scripts PHP usan error_reporting durante desarrollo para detectar fallos, pero se oculta la salida de errores (display_errors=0) en producción.
- Se recomienda, para producción, deshabilitar reportes de errores en pantalla y guardar logs en archivos separados.

9.3. Desarrollo multiplataforma (App Android)

9.3.1. Descripción general

- La aplicación se divide en tres actividades principales:
    1. **MainActivity**: visualización del catálogo y resumen de pedido.
    2. **AddProductActivity**: formulario para alta y posible inserción en almacén.
    3. **StoreActivity**: gestión de inventario y alertas.
- La comunicación con el servidor se realiza mediante llamadas asíncronas de Retrofit para no bloquear el hilo principal.
- Se usa **Network Security Config** que permite tráfico HTTP durante desarrollo.

9.3.2. Asegurar funcionalidad en distintos dispositivos

- Se ha probado la app en Table propia.
- Se han validado comportamientos de DatePicker, RecyclerView y WorkManager
- Los tamaños y márgenes en layouts están basados en dp y sp, ajustando bien la interfaz tanto en pantallas grandes como en más pequeña.

---

10. DOCUMENTACIÓN

10.1. Documentación interna de código

10.1.1. Descripción de cada fichero (autor, función, fecha)

1. **RetrofitClient.kt**
    - **Autor**: José Manuel Monsa Jimeno
    - **Función**: Configura Retrofit con base URL.
    - **Fecha**: 15/04/2025
2. **ApiService.kt**
    - **Autor**: José Manuel Monsa Jimeno
    - **Función**: Interfaz que define los endpoints REST del backend.
    - **Fecha**: 15/04/2025
3. **MainActivity.kt**
    - **Autor**: José Manuel Monsa Jimeno
    - **Función**: Muestra el catálogo de productos y agrupa un resumen de pedido.
    - **Fecha**: 28/03/2025
4. **AddProductActivity.kt**
    - **Autor**: José Manuel Monsa Jimeno
    - **Función**: Permite al usuario agregar un producto al catálogo y, opcionalmente, insertar fechas en almacén.
    - **Fecha**: 13/05/2025
5. **StoreActivity.kt**
    - **Autor**: José Manuel Monsa Jimeno

- **Función**: Pantalla de gestión de inventario de almacén; muestra tabla con agrupación y accesos a editar/eliminar/consumir.
- **Fecha**: 28/04/2025

6. **NotificationWorker.kt**
   - **Autor**: José Manuel Monsa Jimeno
   - **Función**: Worker de WorkManager que dispara notificaciones locales dos semanas antes de caducidad.
   - **Fecha**: 17/04/2025
7. **getProductos.php**
   - **Autor**: José Manuel Monsa Jimeno
   - **Función**: Devuelve JSON con productos disponibles, opcionalmente filtrando por marca.
   - **Fecha**: 03/04/2025
8. **getAlmacen.php**
   - **Autor**: José Manuel Monsa Jimeno
   - **Función**: Devuelve JSON con datos de inventario (JOIN entre productos y almacén).
   - **Fecha**: 03/04/2025
9. **addProducto.php**
   - **Autor**: José Manuel Monsa Jimeno
   - **Función**: Inserta producto nuevo o, si existe, devuelve ID para agregar únicamente fechas de almacén.
   - **Fecha**: 13/05/2025
10. **addAlmacen.php**
    - **Autor**: José Manuel Monsa Jimeno
    - **Función**: Inserta en tabla almacen una fila por cada fecha recibida en formato CSV.
    - **Fecha**: 13/05/2025
11. **editProducto.php**
    - **Autor**: José Manuel Monsa Jimeno
    - **Función**: Actualiza datos de un producto y su almacén (inserta/actualiza/elimina filas relacionadas).
    - **Fecha**: 23/05/2025
12. **available.php**
    - **Autor**: José Manuel Monsa Jimeno
    - **Función**: Cambia bandera Disponibles para un producto dado.
    - **Fecha**: 28/05/2025
13. **deleteProducto.php**
    - **Autor**: José Manuel Monsa Jimeno
    - **Función**: Elimina un producto y todas sus filas relacionadas en almacen en una transacción.
    - **Fecha**: 20/05/2025
14. **consumeProduct.php**
    - **Autor**: José Manuel Monsa Jimeno
    - **Función**: Elimina la unidad con fecha de caducidad más próxima y devuelve stock restante y siguiente fecha.
    - **Fecha**: 31/05/2025

10.2. Documentación externa (MANUALES)

10.2.1. Manual de Instalación

1. **Requisitos del servidor (backend)**
   - Apache ≥ 2.4, PHP ≥ 7.4 con extensiones mysqli y json.
   - MySQL ≥ 5.7 o MariaDB equivalente.
2. **Despliegue del backend**
   1. Clonar o copiar la carpeta api/ al directorio raíz del servidor web (p. ej., htdocs/dulceria/api/).
   2. Importar el script SQL tienda.sql en MySQL (crear base de datos tienda y tablas productos, almacen).
   3. Ajustar credenciales de conexión en cada script PHP (host, usuario, contraseña, nombre BD).
3. **Instalación en Android**
   1. Abrir el proyecto en Android Studio.
   2. Ajustar la BASE_URL en RetrofitClient.kt con la IP o dominio del servidor donde se alojan los scripts PHP.
   3. Conectar un dispositivo o emulador con API ≥ 21.
   4. Compilar y ejecutar la aplicación.

10.2.2. Manual de Usuario

1. **Pantalla Principal (MainActivity)**
   - Al iniciar la app, se carga automáticamente la marca “Cordero” con productos disponibles.
   - Para ver otra marca, presionar el icono correspondiente en el scroll superior.
   - Para filtrar por tamaño, pulsar “Pequeño” o “Grande”.
   - Al seleccionar un producto, se muestra su nombre y precio a la derecha.
   - Para añadir al resumen de pedido, tocar sobre la tarjeta del producto. Los botones “–” y “+” permiten ajustar la cantidad por producto, y el total global se actualiza en tiempo real.
   - El botón “Nuevo Pedido” limpia el resumen.
   - El botón “Almacén” abre la pantalla de gestión de inventario.
   - El botón “Añadir Producto” abre el formulario de alta.
2. **Formulario “Añadir Producto” (AddProductActivity)**
   - Rellenar “Nombre del producto”, “Tamaño” (auto-completable), “Precio (€)” y “Marca” (auto-completable).
   - Marcar “Disponibles” si se desea que el producto salga en el catálogo.
   - Marcar “Almacenar” para añadir fechas de caducidad: aparecerán un NumberPicker y un conjunto de campos “Fecha i”.
   - Seleccionar la cantidad (1–10) en NumberPicker; se generarán tantos campos de fecha como cantidad.
   - Pulsar cada campo de fecha para abrir un DatePicker que no permita fechas anteriores a hoy.
   - Presionar “Guardar” para invocar addProducto.php.
     - Si el producto no existía, se añadirá y regresará al listado.

        - Si ya existía, se mostrará un diálogo que pregunta si se desean agregar fechas en almacén. Al pulsar "Sí", se invoca addAlmacen.php con el ID existente.
3. **Pantalla de Inventario (StoreActivity)**
    - Muestra una tabla con columnas: Cantidad, Producto, Marca, Tamaño, Fecha Caducidad.
    - Arriba, se encuentran filtros:
        - **Nombre**: Filtra pornombre.
        - **Marca**: menú emergente que lista "Todas" + marcas existentes.
        - **Fecha**: Abre DatePicker para filtrar fechas de caducidad .
        - **Tamaño**: menú emergente con "Pequeño" / "Grande".
        - **Reiniciar**: limpia todos los filtros y recarga la tabla original.
    - Pulsar sobre la fila abre un Menu con: "Modificar", "Eliminar", "Consumir".
        - **Modificar**: abre un diálogo con campos precargados (producto, tamaño, precio, marca, disponibles, almacen, fecha).
        - **Eliminar**: Borra producto y todas sus unidades de almacén.
        - **Consumir**: Se resta la unidad con caducidad más próxima. Si al consumir quedan 0 unidades, muestra "Producto agotado".
4. **Notificaciones de caducidad**
    - Esto programa envía una notificación local con título "Producto a punto de caducar" y mensaje "El producto <nombre> caduca el <fecha>" se envía con 14 días de antelación antes de caducarse.

10.2.3. Documentación API

- **GET /api/getProductos.php?marca={marca}**
    - **Descripción**: Devuelve lista de productos con Disponibles = 1; si se especifica marca, filtra por ese valor.
    - **Parámetros**:
        - marca (opcional, String)
    - **Respuesta** :

      [{"ID":"1","Producto":"Agua","Tamano":"Grande","Precio":"1.50","Marca":"Cordero","Disponibles":"1","Almacen":"1"}]

- **GET /api/getAlmacen.php**
    - **Descripción**: Devuelve lista completa de registros de inventario (JOIN con productos) para aquellos con Almacen = 1.
    - **Respuesta** (200 OK):

      [{"ID":29,"Producto":"Agua","Tamano":"Pequeño","Precio":1,"Marca":"Cordero","Disponibles":1,"Almacen":1,"Cantidad":2,"FechaCaducidad":"20\/05\/2025"}]

- **POST /api/addProducto.php**
    - **Descripción**: Inserta nuevo producto en tabla productos. Si el producto ya existe, devuelve status="exists".
    - **Cuerpo** (ejemplo):

[{"producto": "Gominolas de Fruta","tamano": "Pequeño","precio": 0.75,"marca","Fini","disponibles": 1,"almacen": 1,"cantidad": 3,"fechaCaducidad": "15/07/2025,16/07/2025,17/07/2025"}]

- **Respuestas**:
  - **200 OK** (nuevo):

    [{ "status": "ok", "newId": 45 }]

  - **200 OK** (ya existe):

    [{ "status": "exists", "existingId": 12 }]

  - **4xx/5xx**: Error con {"status": "error", "message": "Descripción"}.

- **POST /api/addAlmacen.php**
  - **Descripción**: Añade filas en almacen para un producto existente.
  - **Cuerpo**:

    [{"producto_id": 12,"cantidad": 3,"fechaCaducidad": "15/07/2025,16/07/2025,17/07/2025"}]

  - **Respuesta**:
    - **200 OK**: {"status": "ok", "message": "Almacén actualizado para producto_id=12"}
- **POST /api/editProducto.php**
  - **Descripción**: Actualiza los datos de un producto y su almacén.
  - **Campos obligatorios**:
    - id: ID del producto a actualizar
    - producto, tamano, precio, marca
    - disponibles: 0 o 1
    - almacen: 0 o 1
    - Si almacen=1: cantidad (≥ 1) y fechaCaducidad (string “dd/mm/yyyy”).
  - **Respuesta**: { "status": "ok" } o { "status": "error", "message": "…" }.
- **POST /api/available.php**
  - **Descripción**: Marca como disponibles=0/1 un producto dado su ID.
  - **Campos**: id (int), disponibles (0 o 1)
  - **Respuesta**: { "status": "ok" } o { "status": "error", "message": "…" }.
- **POST /api/deleteProducto.php**
  - **Descripción**: Elimina un producto y sus unidades en almacen.
  - **Campos**: id (int)
  - **Respuesta**: { "status": "ok" } o { "status": "error", "message": "…" }.
- **POST /api/consumeProduct.php**
  - **Descripción**: Elimina la unidad con fecha de caducidad más próxima de un producto.
  - **Campos**: id (int)
  - **Respuesta**:

    [{ "status": "ok", "newCount": 4, "nextDate": "20/08/2025" }]

    o, si no quedan unidades:

[{ "status": "ok", "newCount": 0, "nextDate": null }]

---

11. DESPLIEGUE

11.1. Descripción de la instalación o despliegue

1. **Servidor Lámpara (LAMP)**
   - **Sistema Operativo**: Linux (Ubuntu Server 20.04 recomendado)
   - **Apache**: Configurar VirtualHost para que apunte a la carpeta /api.
   - **PHP**: Habilitar extensiones mysqli y json.
   - **MySQL**: Crear usuario y contraseña dedicado (no root).
   - **Despliegue**:
     1. Copiar scripts PHP a /var/www/html/dulceria/api/.
     2. Importar fichero SQL (tienda.sql)
     3. Ajustar permisos de carpeta para que Apache pueda leer.
     4. (Opcional) Configurar entorno HTTPS: instalar certificado SSL (Let's Encrypt) y forzar redirección HTTP→HTTPS.
2. **Aplicación Android**
   - Descargar el APK generado en Android Studio (Build → Build APK).
   - Si se instala en un dispositivo físico fuera del entorno de desarrollo, habilitar "Instalación desde orígenes desconocidos".
   - Configurar en ajustes RetrofitClient.BASE_URL con la IP o dominio público del servidor.
   - Abrir la app → Comprobar conexión con el servidor (se muestra catálogo de productos).

---

12. HERRAMIENTAS DE APOYO

- **Entorno de desarrollo backend**:
  - XAMPP (para pruebas locales), PHPStorm o Visual Studio Code (con extensiones PHP).
  - MySQL Workbench (para diseñar DER y ejecutar consultas).
- **Entorno de desarrollo Android**:
  - Android, con SDK de Android, Kotlin plugin.
- **Control de versiones**:
  - Git (https://github.com/jmmj2712/TFG.kt).
- **Diagramas y documentación**:
  - Microsoft Word
  - JavaDoc.
- **Pruebas**:
  - Pruebas en Tables

---

13. CONTROL DE VERSIONES

- Se ha utilizado **Git** durante el desarrollo (https://github.com/jmmj2712/TFG.kt)
- Estructura de ramas:
  - main (o master): versión estable y última entrega de TFG.
- Repositorio privado en GitHub con historial completo de commits.

---

14. SISTEMAS DE INTEGRACIÓN CONTINUA

- En este proyecto no se ha implementado un sistema integrado continuo, pero se recomienda en futuras iteraciones:
  - **GitHub Actions** para:
    - Ejecutar lint de Kotlin.
    - Ejecutar pruebas unitarias.
    - Verificar formato de código PHP.

---

15. GESTIÓN DE PRUEBAS

15.1. Estrategia de pruebas

- **Pruebas funcionales manuales**: se ejecutaron en Table y servidor local XAMPP para verificar:
  1. Listado de productos por marca y tamaño (MainActivity).
  2. Alta de producto con y sin almacén (AddProductActivity).
  3. Consumo de unidades en almacén (StoreActivity).
  4. Notificaciones en Android cuando la fecha de caducidad estaba a menos de 14 días.
  5. Edición y eliminación de productos y de unidades de almacén.
- **Pruebas de API con Postman**:
  - GET /getProductos.php?marca=
  - POST /addProducto.php con JSON válido/erróneo.
  - POST /addAlmacen.php con CSV de fechas.
  - POST /editProducto.php, /available.php, /deleteProducto.php y /consumeProduct.php comprobando respuestas esperadas y manejo de errores.

---

16. CONCLUSIONES

16.1. Grado de cumplimiento de objetivos

- Se ha implementado con éxito una solución completa que cumple con los requisitos principales:
  - Gestión de catálogo de productos (alta, edición, eliminación).
  - Inventario con fechas de caducidad por unidad.
  - Notificaciones automáticas dos semanas antes de caducidad.

  - Interfaz table usable en Android con filtros por marca y tamaño.
- **Aspectos cubiertos**:
  - Base de datos relacional con integridad referencial.
  - Backend en PHP con consultas seguras (parametrizadas).
  - Cliente Android con Retrofit, RecyclerView y WorkManager.
  - Documentación técnica interna y externa (manuales de instalación y usuario).

16.2. Propuestas de ampliación

1. **Módulo de facturación y ventas**
   - Incorporar un módulo POS para registrar ventas, generar tickets y actualizar stock automáticamente.
2. **Panel web administrativo**
   - Desarrollar una interfaz web (Angular, React o PHP + Blade) para gestionar productos e inventario desde navegador.
3. **Incorporar escáner de códigos de barras**
   - Para acelerar la introducción de productos, se plantea usar un lector de código de barras que rellene automáticamente los datos, haciendo el alta más ágil y sencilla.
4. **Gestión avanzada de proveedores y compras**
   - Añadir un módulo para controlar las compras a proveedores, identificar productos más económicos según proveedor, y obtener estadísticas de productos más vendidos, facilitando decisiones de reposición y negociación de precios.

---

17. BIBLIOGRAFÍA

1. Lucidchart. https://www.lucidchart.com/
2. Google. (2022). WorkManager Overview. Recuperado de https://developer.android.com/topic/libraries/architecture/workmanager
3. MySQL Documentation. (s. f.). MySQL 8.0 Reference Manual. www.mysql.com
4. PHP: The Right Way. (s. f.). Recuperado de https://phptherightway.com/