# EXAMEN PRÁCTICO - INDICADOR N°1

Sistema de Gestión de Inventario de Biblioteca

2025

#### Año de la Recuperación y Consolidación de la Economía Peruana

- **Proyecto**: Sistema de Gestión de Inventario de Biblioteca
- Alumno: Castillo Rijalba Kimberly Briana
- Curso: Desarrollo de Aplicaciones Móviles
- **Docente**: ING. Omar Burgos Palacios
- **Programa de Estudios**: Desarrollo de Sistemas de Información Ciclo IV

#### Introducción

#### Objetivos del Proyecto

El objetivo principal de este proyecto fue **diseñar** e **implementar** una aplicación funcional para la gestión completa de inventario, aplicando el paradigma **CRUD** (Crear, Leer, Actualizar, Eliminar). Se buscó establecer una comunicación robusta y eficiente entre una interfaz de usuario moderna (**Flutter**) y una base de datos relacional (**MySQL**) a través de un **API RESTful**.

#### Alcance del Sistema

El alcance del sistema se limita a la **administración de la entidad productos**. El sistema permite a los usuarios:

- Visualizar un listado completo de productos (lista\_productos.dart).
- Registrar nuevos productos (agregar\_productos.dart).
- Ver el detalle de un producto específico (detalle\_producto.dart).
- Modificar los datos de un producto existente (editar\_productos.dart).
- Eliminar productos del inventario.

El proyecto se centra en la **funcionalidad transaccional** (conexión, envío y recepción de datos) y la **usabilidad básica** de la interfaz.

## Análisis de requerimientos

#### Requerimientos Funcionales

• **Gestión de Listado:** El sistema debe mostrar un listado dinámico de todos los productos disponibles, con indicadores visuales para el bajo stock.

- **Registro (C)**: El sistema debe permitir al usuario ingresar datos de un nuevo producto (Nombre, Precio, Stock, etc.) y registrarlo en la base de datos.
- Consulta (R): El sistema debe permitir visualizar los detalles completos de un producto seleccionado.
- **Edición (U):** El sistema debe permitir modificar los datos de un producto existente, manteniendo su identificador original (id).
- Eliminación (D): El sistema debe permitir la eliminación permanente de un producto, previa confirmación.
- API Response: La API debe responder a todas las peticiones con un formato JSON estandarizado ({"success": true/false, "message": "..."}).

#### Requerimientos NO Funcionales

- Usabilidad: La interfaz debe ser intuitiva, con diseño moderno (Material Design) y campos de formularios claros.
- **Rendimiento**: El tiempo de respuesta de las peticiones CRUD no debe superar los 2 segundos en un entorno local.
- Compatibilidad: La aplicación debe ser funcional en la plataforma Web
   (navegador Chrome) y potencialmente en Android.
- Seguridad (CORS): La API de PHP debe implementar los encabezados de CORS (Access-Control-Allow-Origin: \*) para permitir la comunicación con el frontend de Flutter.

#### Diseño del Sistema

#### Diseño de la Arquitectura de Capas

El proyecto se basa en una arquitectura de tres capas, utilizando tecnologías asíncronas para la comunicación:

- Capa de Presentación (Frontend): Desarrollada en Flutter, contiene las pantallas de la aplicación. El InventarioService se encarga de realizar las peticiones HTTP.
- Capa de Lógica/Servicio (Backend): Implementada en PHP, aloja la lógica de ruteo y las funciones CRUD (api.php). Resuelve las peticiones HTTP y se comunica con la base de datos.
- Capa de Datos (Base de Datos): MySQL (biblioteca\_flutter) que almacena la información.

#### Diagrama Entidad-Relacion (Base de Datos)

El sistema utiliza una única tabla principal, productos.



# **Implementación**

#### Conectividad y Solución Técnica

La comunicación se centralizó en el archivo **inventario\_service.dart**, utilizando la librería http. La principal solución técnica fue establecer la URL base para el entorno de desarrollo local, incluyendo el puerto no estándar de Apache:

```
class InventarioService {
    // URL FINAL CORREGIDA: Usa el puerto 8080 para la ejecución en web (navegador)
    final String _baseUrl = 'http://localhost:8080/API_INVENTARIO/api.php';
```

#### Explicación del Código CRUD

Las operaciones de **Actualizar** y **Eliminar** se implementaron asegurando el envío correcto de la clave principal (id):

- Edición (actualizarProducto): Se envía el objeto Producto completo,
   incluyendo el id, para que el API PHP lo use en la cláusula WHERE del UPDATE.
- Eliminación (eliminarProducto): Se envía solo el id encapsulado en un objeto JSON.

#### Implementación de Diseño

Se utilizaron los *widgets* Card, ListTile y estilos de InputDecoration avanzados para modernizar la interfaz:

```
child: Card(

// Establecer un color sutil como fondo si hay poco stock

color: lowStock ? lowStockColor: Theme.of(context).cardColor,

elevation: 2,

shape: RoundedRectangleBorder(

borderRadius: BorderRadius.circular(10), // Bordes redondeados

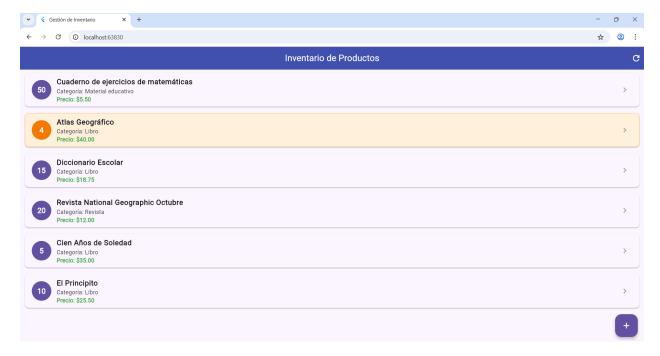
side: lowStock ? BorderSide(color: Colors.orange.Shade200, width: 1): BorderSide.none,

), // RoundedRectangleBorder
```

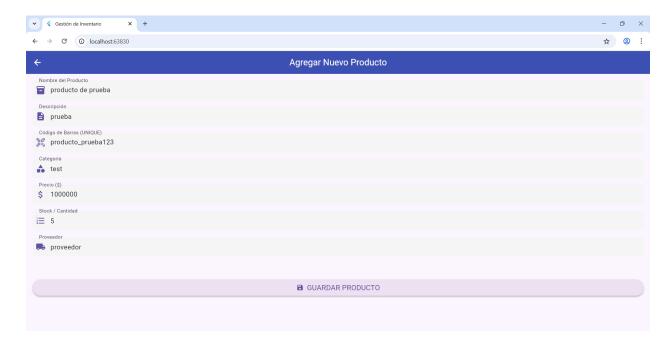
# **Pruebas y Evidencias**

Las pruebas se realizaron de forma manual, validando cada una de las operaciones CRUD y la persistencia de los datos en MySQL.

La lista se carga sin errores de conexión.



• Creación de un nuevo producto



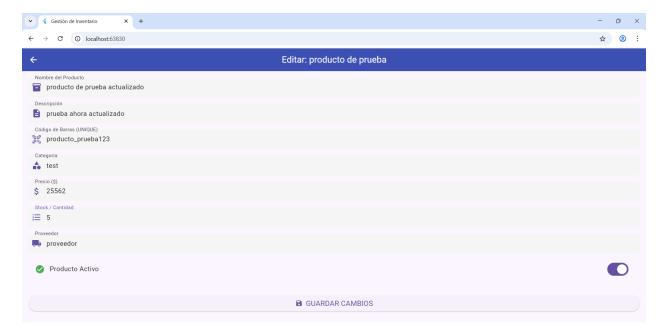
• Producto creado exitosamente. Se logra visualizar en la parte inferior un mensaje emergente



 Se logra visualizar la creación del nuevo registro (producto) en la base de datos:



• Modificación de un producto:



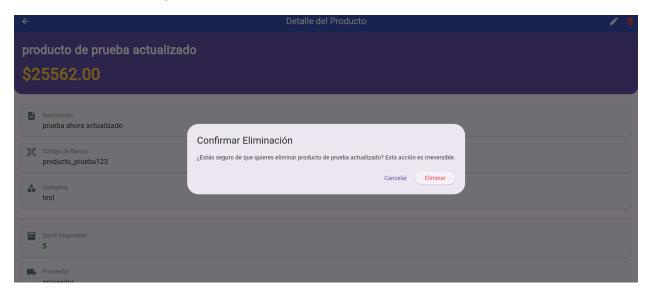
 Producto editado exitosamente: Se muestra una ventana emergente mostrando la confirmación de la actualización del producto



Se visualiza el registro de la base de datos actualizado



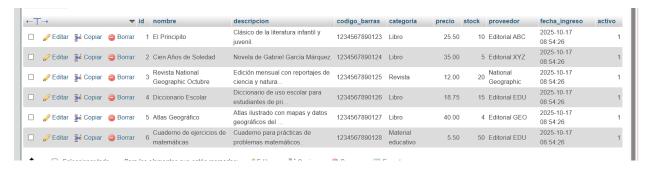
• Eliminación de un producto. Pide confirmar la eliminación para evitar eliminaciones por error



• Producto eliminado con éxito. Se muestra una ventana emergente



• Desaparece el registro de la base de datos



Conexión exitosa con JSON

# 

### **Conclusiones**

El proyecto demostró la viabilidad y la robustez de una arquitectura de desarrollo Full Stack, combinando la flexibilidad de **Flutter** para la interfaz de usuario con la estabilidad de **PHP/MySQL** para la capa de servicio y datos.

La implementación de las operaciones CRUD fue exitosa. La principal dificultad técnica fue la **resolución del conflicto de puertos de red** (IIS vs XAMPP), que requirió ajustes específicos en la configuración del servidor local y en el código del InventarioService. Superar este obstáculo fue crucial para establecer una comunicación persistente y confiable.

El resultado final es un sistema de inventario **funcional, escalable en su arquitectura y con una interfaz de usuario mejorada**, listo para ser expandido con características como autenticación y filtrado avanzado. El proceso consolidó la comprensión práctica de las peticiones HTTP, el manejo de JSON y la arquitectura de desarrollo en capas.