LIBRERIA FANTASIA

MANUAL TECNICO

ÍNDICE

INTRODUCCION

Este manual técnico proporciona una descripción exhaustiva del sistema de gestión de librería, una aplicación de escritorio robusta y fácil de usar, desarrollada utilizando el lenguaje de programación Python y el potente framework PyQt6 para la interfaz gráfica. El propósito fundamental de este sistema es simplificar y optimizar todas las operaciones diarias que se llevan a cabo en una librería. Esto incluye, pero no se limita a, la gestión eficiente de los usuarios del sistema, el registro y seguimiento de todas las ventas, la administración de las compras a proveedores, el control detallado del inventario de productos y la generación de reportes clave para la toma de decisiones.

Nuestro objetivo al crear este sistema es ofrecer una herramienta confiable y accesible que permita a cualquier miembro del personal de la librería, desde los administradores con acceso total hasta los vendedores con permisos específicos, manejar de forma eficiente y sin complicaciones los procesos esenciales del negocio. Esto se traduce en una mayor productividad, una mejor organización de los recursos y una visión clara del rendimiento de la librería.

ARQUITECTURA DEL SISTEMA

El sistema de gestión de librería ha sido diseñado siguiendo una arquitectura modular. Esto significa que el software está dividido en componentes más pequeños e independientes, cada uno con una función específica. Esta división no es arbitraria; se realiza con el propósito de facilitar enormemente el desarrollo (ya que diferentes partes pueden trabajarse en paralelo), el mantenimiento (es más fácil encontrar y corregir errores en un módulo específico) y la escalabilidad (se pueden añadir nuevas funcionalidades o mejorar las existentes sin afectar todo el sistema). Esta estructura bien definida asegura que cada parte del sistema tenga un propósito claro y una responsabilidad limitada, lo que simplifica la identificación y resolución de problemas, así como la implementación de nuevas funcionalidades en el futuro.

CAPA DE PRESENTACIÓN (UI - USER INTERFACE):

- ¿Qué es? Esta es la parte del sistema con la que el usuario final interactúa directamente. Es lo que ves y con lo que haces clic.

- ¿Cómo funciona? Está construida utilizando el framework PyQt6, que permite crear ventanas, botones, campos de texto, tablas y otros elementos visuales. Se encarga de mostrar toda la información de manera clara y organizada, y de recibir las acciones que el usuario realiza (por ejemplo, escribir en un campo, hacer clic en un botón)

- Organización: Cada sección principal de la aplicación, como la gestión de usuarios, el punto de venta para registrar ventas, la sección de compras, la gestión del inventario o la generación de reportes, tiene su propia "ventana" o interfaz dedicada. Esto ayuda a organizar la experiencia del usuario, haciendo que la navegación sea intuitiva y que las tareas se agrupen lógicamente.

- Ejemplos: La ventana de inicio de sesión, la tabla que muestra el inventario de productos, los formularios para agregar un nuevo usuario, los botones para confirmar una venta.

ARQUITECTURA DEL SISTEMA

CAPA DE LÓGICA DE NEGOCIO:

- ¿Qué es? Esta capa puede considerarse el "cerebro" del sistema. Es donde residen las reglas y procesos que definen cómo funciona la librería.

- ¿Cómo funciona? Se encuentra implementada dentro de las clases de las ventanas de la interfaz de usuario (UI). Su función principal es tomar las solicitudes del usuario (recibidas por la UI), procesarlas, aplicar las reglas de negocio específicas de la librería (por ejemplo, "no se puede vender un producto si no hay stock disponible", "calcular el total de una compra sumando los costos de los productos"), y asegurarse de que todos los datos sean válidos y consistentes antes de que se intenten guardar.

- Responsabilidades Adicionales: También se encarga de tareas más complejas como la generación de documentos PDF para recibos de venta, órdenes de compra o reportes detallados, asegurando que la información presentada sea precisa y cumpla con los requisitos del negocio.

- Ejemplos: La lógica que verifica si un usuario tiene permiso para acceder a una sección, el cálculo automático del total de una venta o compra, la validación de que un campo de texto solo contenga números.

CAPA DE ACCESO A DATOS (DAL - DATA ACCESS LAYER):

- ¿Qué es? Esta capa actúa como un puente vital entre la aplicación y la base de datos donde se almacena toda la información.
- ¿Cómo funciona? Está centralizada en una única clase llamada BaseDatos. Su trabajo exclusivo es manejar toda la comunicación con el sistema de gestión de bases de datos MySQL. Esto incluye operaciones fundamentales como:
- Crear (C): Guardar nueva información (ej. un nuevo usuario, un nuevo producto).
- Leer (R): Recuperar datos existentes de la base de datos (ej. obtener la lista de todos los productos, buscar un usuario por su nombre).
- Actualizar (U): Modificar registros existentes (ej. cambiar el precio de un producto, actualizar el stock después de una venta).
- Eliminar (D): Borrar información (ej. eliminar un proveedor).

ARQUITECTURA DEL SISTEMA

CAPA DE ACCESO A DATOS (DAL - DATA ACCESS LAYER):

Beneficio: Gracias a esta capa, el resto de la aplicación (la UI y la lógica de negocio) no necesita saber los detalles técnicos de cómo se almacenan los datos o cómo se interactúa con MySQL. Simplemente "pide" o "envía" información a la capa DAL, y esta se encarga de la complejidad subyacente. Esto hace que el sistema sea más flexible; si en el futuro se decide cambiar a otro tipo de base de datos, solo esta capa necesitaría ser modificada.

CAPA DE UTILIDADES:

- ¿Qué es? La clase Codigo es una clase base que centraliza un conjunto de funciones y herramientas que son útiles en múltiples partes de la aplicación.

- ¿Cómo funciona? Proporciona funcionalidades comunes que se reutilizan en todas las ventanas y módulos. Esto incluye la aplicación de estilos visuales a los botones y otros elementos de la interfaz (para mantener una apariencia uniforme), métodos para limpiar el contenido de los layouts (organización de elementos en pantalla) y funciones estandarizadas para mostrar mensajes de información, advertencia o error al usuario.

- Beneficio: Al reutilizar estas utilidades, se garantiza una apariencia y un comportamiento consistentes en todo el sistema, se reduce la cantidad de código duplicado y se facilita la modificación global de estilos o comportamientos.

CONFIGURACIÓN E INSTALACIÓN

Para que el sistema de gestión de librería funcione correctamente en su entorno, es necesario cumplir con ciertos requisitos y seguir los pasos de instalación y configuración.

REQUISITOS DEL SISTEMA

Antes de instalar el software, asegúrese de que su sistema cumple con los siguientes requisitos:

- Python 3.x: El sistema está desarrollado en Python. Es fundamental tener una versión de Python 3.x instalada en su máquina. Se recomienda encarecidamente utilizar la versión más reciente y estable disponible para asegurar la compatibilidad y aprovechar las últimas mejoras de rendimiento y seguridad. Puede descargarla desde el sitio web oficial de Python.

- Bibliotecas de Python: El sistema depende de varias bibliotecas externas que extienden las funcionalidades de Python. Estas son como "paquetes" de código preescrito que el sistema utiliza para tareas específicas:
  - PyQt6: Esta es la biblioteca principal para la interfaz gráfica de usuario. Es la herramienta que permite crear todas las ventanas, botones, tablas y otros elementos visuales que componen la aplicación. Sin PyQt6, la aplicación no podría mostrarse.
  - PyMySQL: Esta biblioteca es esencial para la comunicación con la base de datos. Permite que la aplicación se conecte a un servidor MySQL y realice operaciones de lectura, escritura, actualización y eliminación de datos.
  - bcrypt: La seguridad es primordial. bcrypt es una biblioteca criptográfica que se utiliza específicamente para el hashing de contraseñas. Esto significa que las contraseñas de los usuarios no se almacenan en texto plano en la base de datos, sino que se transforman en una cadena de caracteres ilegible (un "hash"). Esto protege la información del usuario en caso de una brecha de seguridad.
  - reportlab: Para la generación de documentos profesionales, el sistema utiliza reportlab. Esta biblioteca es la encargada de crear los documentos en formato PDF, como los recibos de venta para los clientes o los reportes detallados para la administración.

- Servidor de Base de Datos: Se requiere un servidor MySQL (o un sistema de gestión de bases de datos compatible con MySQL). Aquí es donde toda la información operativa de la librería (usuarios, productos, ventas, etc.) se almacenará de forma persistente y organizada. Asegúrese de tener acceso a un servidor MySQL y las credenciales necesarias (nombre de usuario y contraseña) para crear bases de datos y tablas.

CONFIGURACIÓN E INSTALACIÓN

INSTALACIÓN DE DEPENDENCIAS

Antes de instalar el software, asegúrese de que su sistema cumple con los siguientes requisitos:

- Python 3.x: El sistema está desarrollado en Python. Es fundamental tener una versión de Python 3.x instalada en su máquina. Se recomienda encarecidamente utilizar la versión más reciente y estable disponible para asegurar la compatibilidad y aprovechar las últimas mejoras de rendimiento y seguridad. Puede descargarla desde el sitio web oficial de Python.

Una vez que Python esté instalado, el siguiente paso es instalar las bibliotecas de Python mencionadas. Para ello, necesitará una conexión a internet activa. Abra una terminal o línea de comandos (en Windows, puede ser cmd o PowerShell; en macOS/Linux, el Terminal) y ejecute el siguiente comando:
pip install PyQt6 PyMySQL bcrypt reportlab

Recomendación Importante: Entornos Virtuales Para una gestión limpia y evitar conflictos entre diferentes proyectos de Python, es altamente recomendable utilizar un entorno virtual. Un entorno virtual crea un espacio aislado para las dependencias de su proyecto.

Pasos para usar un entorno virtual (opcional pero recomendado):

1. Crear el entorno virtual: python -m venv venv
2. (Esto creará una carpeta venv en su directorio de proyecto).
3. Activar el entorno virtual:
   - Windows: .\venv\Scripts\activate
   - macOS/Linux: source venv/bin/activate Una vez activado, verá (venv) al principio de su línea de comandos.
4. Instalar dependencias dentro del entorno activado: pip install PyQt6 PyMySQL bcrypt reportlab

De esta manera, las bibliotecas se instalarán solo para este proyecto, sin afectar otras instalaciones de Python en su sistema.

- Bibliotecas de Python: El sistema depende de varias bibliotecas externas que extienden las funcionalidades de Python. Estas son como "paquetes" de código preescrito que el sistema utiliza para tareas específicas:
  - PyQt6: Esta es la biblioteca principal para la interfaz gráfica de usuario. Es la herramienta que permite crear todas las ventanas, botones, tablas y otros elementos visuales que componen la aplicación. Sin PyQt6, la aplicación no podría mostrarse.
  - PyMySQL: Esta biblioteca es esencial para la comunicación con la base de datos. Permite que la aplicación se conecte a un servidor MySQL y realice operaciones de lectura, escritura, actualización y eliminación de datos.

CONFIGURACIÓN E INSTALACIÓN

INSTALACIÓN DE DEPENDENCIAS

- bcrypt: La seguridad es primordial. bcrypt es una biblioteca criptográfica que se utiliza específicamente para el hashing de contraseñas. Esto significa que las contraseñas de los usuarios no se almacenan en texto plano en la base de datos, sino que se transforman en una cadena de caracteres ilegible (un "hash"). Esto protege la información del usuario en caso de una brecha de seguridad.

- reportlab: Para la generación de documentos profesionales, el sistema utiliza reportlab. Esta biblioteca es la encargada de crear los documentos en formato PDF, como los recibos de venta para los clientes o los reportes detallados para la administración.

- Servidor de Base de Datos: Se requiere un servidor MySQL (o un sistema de gestión de bases de datos compatible con MySQL). Aquí es donde toda la información operativa de la librería (usuarios, productos, ventas, etc.) se almacenará de forma persistente y organizada. Asegúrese de tener acceso a un servidor MySQL y las credenciales necesarias (nombre de usuario y contraseña) para crear bases de datos y tablas.

CONFIGURACIÓN DE LA BASE DE DATOS

El corazón del sistema de gestión de librería reside en su base de datos MySQL. Es crucial entender cómo se organiza esta información para un mantenimiento adecuado. No es necesario ser un experto en SQL, pero una comprensión básica de la estructura de las "tablas" le será de gran ayuda.

- Organización de la Información en la Base de Datos (Tablas):
  - Imagine la base de datos como un archivador grande y muy bien organizado, donde cada "carpeta" dentro de ese archivador es lo que llamamos una tabla. Cada tabla guarda un tipo específico de información, y dentro de cada tabla, la información se organiza en filas (registros individuales) y columnas (campos específicos de datos).

ESTRUCTURA DE ARCHIVOS Y MÓDULOS

El proyecto está cuidadosamente organizado en varios archivos Python, cada uno con una responsabilidad clara y específica. Esta modularidad facilita la comprensión, el mantenimiento y la colaboración en el código.

- main.py:
  - Rol: Es el punto de entrada principal de la aplicación. Cuando ejecuta el sistema, este es el primer archivo que se ejecuta.
  - Contenido: Se encarga de inicializar la aplicación PyQt6 (QApplication) y de crear y mostrar la primera ventana del sistema, que es la Ventana_inicio (la pantalla de inicio de sesión). También maneja el ciclo de vida de la aplicación.
- Base_datos.py:
  - Rol: Contiene la clase BaseDatos, que es la única responsable de interactuar con la base de datos MySQL. Actúa como un intermediario entre la lógica de la aplicación y el almacenamiento de datos.
  - Contenido: Dentro de esta clase, encontrará métodos para:
    - Establecer y gestionar la conexión con MySQL.
    - Realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para todas las tablas del sistema (usuarios, productos, ventas, compras, proveedores, detalles de venta y compra).
    - Ejecutar consultas SQL para obtener información específica (ej. obtener_productos(), buscar_usuario(), actualizar_stock()).
    - Manejar la inserción de datos, asegurando que se haga de forma segura para prevenir inyecciones SQL.
- codigo.py:
  - Rol: Define la clase Codigo, que sirve como una clase base de utilidades para todas las demás ventanas de la aplicación. Su propósito es centralizar funcionalidades comunes y estilos visuales.
  - Contenido: Incluye métodos genéricos como:
    - imagen(): Para cargar y escalar imágenes para iconos o logos.
    - fondo_degradado(): Para aplicar estilos de fondo con degradados a las ventanas.
    - color_boton_sin_oprimir(), color_boton_oprimido(), color_boton__bloqueado(): Para aplicar estilos visuales a los botones en diferentes estados.
    - color_linea(): Para estilizar los campos de entrada de texto (QLineEdit).
    - color_tabla(): Para aplicar estilos a las tablas de datos (QTableWidget).
    - limpieza_layout(): Una función crucial para limpiar los layouts de las ventanas antes de cargar nuevas interfaces, evitando la superposición de elementos.
  - Beneficio: Al heredar de Codigo, todas las ventanas de la UI obtienen estas funcionalidades automáticamente, promoviendo la consistencia en la interfaz de usuario, reduciendo la duplicación de código y facilitando el mantenimiento.

ESTRUCTURA DE ARCHIVOS Y MÓDULOS

- ventana_inicio.py:
- Rol: Implementa la clase Ventana_inicio, que es la primera pantalla que ve el usuario y se encarga de la autenticación.
- Contenido:
  - Interfaz para que el usuario ingrese su nombre de usuario y contraseña.
  - Lógica para verificar las credenciales contra la base de datos (BaseDatos).
  - Uso de bcrypt para comparar la contraseña ingresada con el hash almacenado.
  - Manejo de la transición a la Ventana_principal si la autenticación es exitosa, o mostrar mensajes de error si falla.
  - Funcionalidad para cerrar la aplicación.
- ventana_principal.py:
- Rol: Contiene la clase Ventana_principal, que actúa como el menú principal o "dashboard" de la aplicación una vez que el usuario ha iniciado sesión.
- Contenido:
  - Muestra un conjunto de botones (Usuarios, Ventas, Compras, Inventario, Reportes) que permiten al usuario navegar a las diferentes secciones del sistema.
  - Implementa la lógica para el control de acceso: los botones se habilitan o deshabilitan según el nivel de usuario (Administrador o Vendedor) obtenido de la base de datos.
  - Maneja la funcionalidad de "cerrar sesión", que devuelve al usuario a la Ventana_inicio.
  - Instancia las clases de las otras ventanas (Usuarios, Ventas, Compras, Inventario, Reportes) para poder mostrarlas cuando el usuario las selecciona.
- ventana_usuarios.py:
- Rol: Gestiona la clase Ventana_usuarios, dedicada a las operaciones de administración de usuarios del sistema.
- Contenido:
  - Interfaz para visualizar una lista de todos los usuarios registrados en una tabla.
  - Funcionalidades CRUD (Crear, Leer, Actualizar, Eliminar) para los usuarios:
    - Agregar nuevos usuarios con sus roles y contraseñas (hasheadas).
    - Editar la información de usuarios existentes (nombre, email, tipo, estado, contraseña).
    - Eliminar lógicamente usuarios (cambiar su estado a inactivo) o eliminarlos físicamente de la base de datos.
  - Implementa la lógica para buscar usuarios por nombre o ID.
  - Maneja la validación de los campos de entrada para la creación y edición de usuarios.

ESTRUCTURA DE ARCHIVOS Y MÓDULOS

- ventana_ventas.py:
- Rol: Implementa la clase Ventana_ventas, que es el punto de venta (POS) de la librería.
- Contenido:
  - Interfaz para que los vendedores registren nuevas ventas.
  - Funcionalidad de "carrito de compras" donde se añaden productos y cantidades.
  - Cálculo automático del total de la venta.
  - Interacción con la base de datos para obtener información de productos y actualizar el stock después de una venta.
  - Generación de recibos de venta en formato PDF utilizando reportlab.
  - Funcionalidad para buscar productos por nombre o ID.
- ventana_compras.py:
- Rol: Contiene la clase Ventana_compras, que se encarga de la gestión de proveedores y las órdenes de compra.
- Contenido:
  - Sección para administrar proveedores: agregar, editar, eliminar y buscar información de proveedores.
  - Sección para crear nuevas órdenes de compra: seleccionar productos del inventario, especificar cantidades y registrar el costo.
  - Funcionalidad para ver las órdenes de compra pendientes.
  - Opción para "confirmar" una orden de compra, lo que implica actualizar el stock de los productos recibidos en el inventario y cambiar el estado de la orden.
  - Generación de reportes de órdenes de compra en PDF.
- ventana_inventario.py:
- Rol: Define la clase Ventana_inventario, responsable de la gestión completa del inventario de productos.
- Contenido:
  - Muestra una tabla detallada de todos los productos en el inventario, incluyendo ID, nombre, descripción, stock actual, precio, costo y stock mínimo.
  - Resalta visualmente los productos cuyo stock está por debajo del stock_minimo.
  - Funcionalidades CRUD para productos: agregar nuevos productos, editar la información de productos existentes y eliminar productos.
  - Lógica para buscar productos por nombre.
  - Actualización de stock (aunque la principal actualización de stock por compras se maneja en ventana_compras).
- ventana_reportes.py:
- Rol: Implementa la clase Ventana_reportes, que permite generar diversos informes para analizar el rendimiento de la librería.

CONSIDERACIONES DE SEGURIDAD

La seguridad es un aspecto fundamental en cualquier sistema que maneje datos, especialmente información de usuarios y transacciones. El sistema de gestión de librería incorpora varias medidas de seguridad:

- Hashing de Contraseñas (bcrypt):
  - Explicación: Cuando un usuario crea una cuenta o cambia su contraseña, el sistema no guarda la contraseña en texto plano (legible) en la base de datos. En su lugar, utiliza la biblioteca bcrypt para aplicar un algoritmo de hashing unidireccional. Esto significa que la contraseña se transforma en una cadena de caracteres aleatoria y fija (el "hash") que es extremadamente difícil de revertir a la contraseña original.
  - Beneficio: Si la base de datos fuera comprometida (por ejemplo, por un ataque informático), los atacantes solo obtendrían los hashes de las contraseñas, no las contraseñas reales. Esto protege las cuentas de los usuarios y minimiza el riesgo de que esas contraseñas sean utilizadas en otros servicios.

- Validación de Entrada de Usuario:
  - Explicación: Antes de procesar o guardar cualquier dato ingresado por el usuario (como nombres de productos, cantidades, precios, etc.), el sistema realiza validaciones. Esto implica verificar que los datos tengan el formato correcto (ej. que un precio sea un número, que una cantidad sea un entero positivo) y que no contengan caracteres inesperados o maliciosos.
  - Prevención de Inyecciones SQL: Aunque PyMySQL ya ofrece una protección significativa al usar parámetros en las consultas (lo cual es una buena práctica implementada en Base_datos.py), la validación adicional a nivel de aplicación reduce aún más el riesgo de ataques de inyección SQL. Una inyección SQL es un tipo de ataque donde un atacante inserta código SQL malicioso en los campos de entrada para manipular la base de datos.

- Control de Acceso Basado en Roles:
  - Explicación: El sistema diferencia entre dos tipos de usuarios principales: 'Administrador' y 'Vendedor'. Cada rol tiene un conjunto predefinido de permisos.
  - Administrador: Tiene acceso completo a todas las funcionalidades del sistema, incluyendo la gestión de usuarios, productos, proveedores, ventas, compras y reportes.
  - Vendedor: Tiene acceso limitado, generalmente solo a las funcionalidades relacionadas con las ventas y posiblemente a la consulta de inventario, pero no puede modificar usuarios, proveedores o configuraciones críticas.
  - Implementación: Esta distinción se maneja en la Ventana_principal, donde los botones para acceder a ciertas secciones se habilitan o deshabilitan dinámicamente según el rol del usuario que ha iniciado sesión. Esto asegura que solo el personal autorizado pueda realizar ciertas operaciones, protegiendo la integridad de los datos y las operaciones de la librería.

MANTENIMIENTO Y SOLUCIÓN DE PROBLEMAS

Esta sección está diseñada para ayudarle a identificar y resolver problemas comunes que puedan surgir durante la operación del sistema, así como para proporcionar pautas sobre cómo mantenerlo y mejorarlo.

- **Error de Conexión a la Base de Datos**:
  - **Síntoma**:
  - La aplicación no logra iniciar, o al intentar realizar cualquier operación que involucre la base de datos (como iniciar sesión o cargar el inventario), muestra un mensaje de error que menciona "MySQL", "conexión", "host", "usuario" o "contraseña".

  - **Causa**:
  - Las credenciales de la base de datos (nombre de usuario, contraseña, dirección del servidor o nombre de la base de datos) configuradas en el archivo Base_datos.py son incorrectas.
  - El servidor MySQL no está en ejecución.
  - Un firewall en su sistema o en la red está bloqueando la conexión al puerto de MySQL (por defecto, 3306).
  - El usuario de MySQL no tiene los permisos adecuados para acceder a la base de datos libreria_db.

  - **Solución**:
  - Verifique el servidor MySQL: Asegúrese de que su servidor MySQL esté activo y corriendo. Puede usar herramientas como MySQL Workbench, phpMyAdmin o los servicios del sistema operativo para verificarlo.
  - Revise Base_datos.py: Abra el archivo Base_datos.py y verifique cuidadosamente que el host, user, password y db (nombre de la base de datos) sean exactamente los mismos que los de su configuración de MySQL.
  - Firewall: Deshabilite temporalmente su firewall para probar la conexión. Si funciona, configure una excepción para el puerto 3306 (o el puerto de MySQL que esté utilizando).
  - Permisos de Usuario MySQL: Asegúrese de que el usuario de MySQL que está utilizando tenga privilegios para SELECT, INSERT, UPDATE, DELETE en la base de datos libreria_db.

MANTENIMIENTO Y SOLUCIÓN DE PROBLEMAS

- **Tablas de Base de Datos Faltantes o Corruptas:**
  - **Síntoma:**
  - La aplicación inicia, pero al intentar acceder a secciones como "Usuarios", "Inventario" o "Ventas", se producen errores que indican que una tabla no existe o que una columna no se encuentra.
  - **Causa:**
  - Las tablas de la base de datos no fueron creadas correctamente (o no se crearon en absoluto) o se eliminaron accidentalmente.
  - **Solución:**
    i. Revisar Esquema: Consulte la sección 3.3 de este manual ("Configuración de la Base de Datos") para obtener el esquema SQL completo de las tablas.
    ii. Ejecutar SQL: Conéctese a su servidor MySQL (usando MySQL Workbench, phpMyAdmin o la línea de comandos) y ejecute los comandos SQL proporcionados en la sección 3.3 para crear todas las tablas necesarias. Asegúrese de seleccionar la base de datos correcta (libreria_db) antes de ejecutar los comandos.
- **Imágenes o Iconos Faltantes:**
  - **Síntoma:**
  - En la interfaz de usuario, algunos botones o etiquetas aparecen vacíos, o se muestra un icono de "imagen rota" en lugar de los iconos esperados (ej. iconos de "agregar", "editar", "eliminar").
  - **Causa:**
  - Las rutas a las imágenes especificadas en el código son incorrectas, o los archivos de imagen no se encuentran en el directorio esperado. El sistema busca las imágenes en una carpeta llamada imagenes/ que debe estar en el mismo directorio donde se ejecutan los archivos Python.
  - **Solución:**
  - Verificar la carpeta imagenes/: Asegúrese de que exista una carpeta llamada imagenes en el mismo nivel que sus archivos .py (ej. main.py, ventana_principal.py).
  - Contenido de la carpeta: Confirme que esta carpeta imagenes contenga todos los archivos de imagen referenciados en el código. Los nombres comunes son usuario.png, logo.ico, logo_libreria.png, inicio.png, inventario.png, usuarios.png, ventas.png, compras.png, reportes.png, anadir.png, editar.png, eliminar.png, agregar.png, buscar.png, proveedores.png, pedido.png, ordenes.png, confirmar.png, detalles.png, infomation.ico, warning.ico.
  - Mayúsculas/Minúsculas: En algunos sistemas operativos (como Linux), los nombres de archivo distinguen entre mayúsculas y minúsculas. Asegúrese de que los nombres de archivo en el código coincidan exactamente con los nombres de los archivos en la carpeta imagenes/.

MANTENIMIENTO Y SOLUCIÓN DE PROBLEMAS

- **Problemas de Rendimiento (Lentitud):**
  - **Síntoma:**
  - La aplicación se siente lenta al cargar datos, especialmente al mostrar tablas con muchos registros (ej. un inventario muy grande, un historial de ventas extenso) o al realizar búsquedas.

  - **Causa**:
  - Las consultas a la base de datos pueden ser ineficientes, o el volumen de datos es muy grande para ser procesado y mostrado rápidamente.

  - **Solución:**
  - Índices en la Base de Datos: Para acelerar las búsquedas y la recuperación de datos, se pueden agregar índices a las columnas de la base de datos que se utilizan frecuentemente en las cláusulas WHERE (búsquedas) o ORDER BY (ordenamiento). Por ejemplo, un índice en la columna nombre de la tabla productos o en fecha_venta de la tabla ventas podría mejorar significativamente el rendimiento. Ejemplo de cómo añadir un índice (ejecutar en su cliente MySQL):
    - CREATE INDEX idx_productos_nombre ON productos (nombre);
    - CREATE INDEX idx_ventas_fecha ON ventas (fecha_venta);
  - Optimización de Consultas: Si una consulta específica es muy lenta, se puede revisar el código SQL en Base_datos.py para buscar formas de optimizarla (ej. evitar SELECT * si solo se necesitan algunas columnas, usar JOIN de manera eficiente).
  - Paginación: Para tablas con un volumen extremadamente grande de datos, se podría implementar la paginación, cargando solo una parte de los datos a la vez en lugar de todos. Esto no está implementado actualmente, pero es una mejora futura.

CONTÁCTANOS

+502 1214 2348 | RedCODE@gmail.com