SISTEMA DE GESTIÓN DE INVENTARIOS PARA UNA FARMACIA

HERRAMIENTAS DE PROGRAMACIÓN II

Eliana Jaramillo

Sammy Arteaga

Catalina Correa

INSTITUCIÓN UNIVERSITARIA PASCUAL BRAVO

3 DE JUNIO DE 2025

**Análisis funcional del sistema**

El sistema de control de inventario para la farmacia fue diseñado para cubrir las necesidades básicas de administración interna, esta parte inicial, está especialmente enfocadas en tres áreas principales: usuarios, productos y proveedores. Este sistema facilita las tareas diarias del personal, ayudando a organizar y mantener actualizada la información clave de la empresa.

Funcionalidades desarrolladas:

1. Gestión de usuarios: El sistema permite registrar nuevos usuarios con un nombre de usuario, contraseña y un rol específico (Administrador, Vendedor o Almacenero). También se puede modificar la información de los usuarios existentes. El objetivo principal de esta funcionalidad es llevar un control de quién tiene acceso al sistema y qué acciones puede realizar según su rol.
2. Gestión de productos: En esta sección, el sistema permite registrar productos con sus respectivos datos: nombre, descripción, precio y cantidad mínima en stock. También se pueden editar o eliminar productos. Esta funcionalidad es esencial para mantener un control organizado del inventario y poder realizar ajustes cuando sea necesario.
3. Gestión de proveedores: Permite guardar y mantener actualizada la información de los proveedores que suministran productos a la farmacia. Aunque por ahora esta funcionalidad se basa principalmente en el registro y mantenimiento de datos, en el futuro se podrá mejorar para relacionar productos con sus proveedores, facilitando la siguimiento de las compras.
4. La funcionalidad de ventas permite registrar de forma sencilla las transacciones realizadas en la farmacia. Para cada venta se seleccionan los productos, se define la cantidad vendida y el sistema calcula automáticamente el total, teniendo en cuenta los precios individuales, los impuestos y los descuentos aplicables. Para facilitar el manejo del dinero y el seguimiento contable, también se genera un comprobante de venta que puede ser impreso o guardado digitalmente. Antes de confirmar la venta, el sistema verifica que haya suficiente stock para cada producto solicitado. Si el inventario no es suficiente, no se permite finalizar la venta.

Cada venta registrada también actualiza automáticamente el inventario, restando la cantidad vendida al stock actual del producto correspondiente.

Flujo general de uso:

* El sistema inicia mostrando los datos actuales (usuarios, productos o proveedores) en un DataGridView.
* El usuario puede seleccionar un registro para modificarlo, o ingresar nuevos datos para registrarlos.
* Los botones de “Guardar”, “Modificar” y “Eliminar” activan las operaciones respectivas. Decidimos no usarlo para eliminar usuarios, ya que en caso de que alguien se retire de la empresa, los datos deben permanecer en la base; porque de lo contrario afectarían significativamente el resto del sistema (por ejemplo, las ventas a su nombre, etc.) y la base de datos.
* El sistema valida los campos antes de guardar cualquier información para evitar errores comunes.

1. Estructura de la base de datos (script SQL)

Para crear la base de datos FarmaciaDB, se usó un script SQL donde se definen todas las tablas que necesita el sistema. Estas tablas permiten guardar toda la información que maneja la farmacia. Por ejemplo:

* La tabla Usuarios guarda los datos de las personas que usan el sistema (como los administradores o empleados).
* La tabla Productos contiene la información de los medicamentos y demás artículos que se venden.
* La tabla Proveedores registra quiénes suministran los productos a la farmacia.

Cada tabla tiene sus columnas, sus llaves primarias, y en algunos casos relaciones entre ellas. Por ejemplo, los productos tienen una relación con los proveedores, ya que en el futuro se podrá saber fácilmente qué proveedor entrega cada producto, lo que va a servir para mejorar los procesos de compras. Este script es muy importante porque automatiza la creación de la estructura de la base de datos y asegura que todos tengamos la misma versión en caso de que varias personas, en nuestro caso somos tres, estén trabajando en el proyecto.

**Justificación técnica de las decisiones tomadas**

Durante el desarrollo de la primera parte de la aplicación, se tomaron varias decisiones técnicas con el fin de lograr que sea clara, mantenible y escalable:

1. Uso del patrón Repository: Se implementó el patrón de diseño Repository para separar la lógica de acceso a los datos del resto de la aplicación. Esto permite que la parte visual (formularios) no esté directamente conectada con la base de datos, sino que se comunique a través de interfaces. Gracias a esto, el código es más limpio y fácil de mantener o modificar en el futuro. Además, facilita las pruebas y permite cambiar la fuente de datos si es necesario (por ejemplo, de una base local a una base de datos en la nube).
2. Diseño de cada formulario: Se creó un formulario diferente para cada funcionalidad principal: usuarios, productos y proveedores; por ahora. Esta organización permite que cada parte del sistema esté separada y no interfiera con las demás. Además, es más sencillo agregar nuevas funcionalidades, como ventas o reportes, sin afectar lo que ya está hecho.
3. Validación desde la interfaz: Antes de guardar datos, se hace una validación en el formulario para asegurarse de que los campos obligatorios estén llenos. Esto asegura la funcionalidad del sistema y evita errores en la base de datos, como campos vacíos o datos mal ingresados.
4. Preparación para crecimiento del sistema: Aunque en esta etapa el sistema solo maneja usuarios, productos y proveedores, su estructura está preparada para crecer; esto gracias al uso de interfaces y del patrón Repository, se podrán agregar otros formularios y clases como facturación, ventas, control de acceso por roles, reportes, etc., sin tener que rehacer lo que ya está funcionando.
5. Patrón Singleton: Conexión con la base de datos (SQL Server): Para que la aplicación pueda guardar y consultar los datos (como los usuarios, productos y proveedores), se necesita una conexión con una base de datos. En este caso usamos SQL Server y creamos una base llamada FarmaciaDB. Para conectarnos a esa base desde el sistema, usamos una clase llamada DatabaseConnection, donde aplicamos un patrón llamado Singleton. Eso lo hicimos para que solo haya una conexión activa a la base de datos durante toda la ejecución de la aplicación, y así no se abran muchas conexiones al mismo tiempo, lo cual podría causar errores o hacer más lento el programa. Esta clase se encarga de abrir la conexión usando una cadena de conexión que está en un archivo de configuración llamado App.config. Ahí es donde se guarda toda la información necesaria para conectarse, como el nombre del servidor, la base de datos, el usuario y la contraseña. Tener eso en un archivo aparte ayuda a que, si se necesita cambiar algo (como la clave o el nombre del servidor), no se tenga que modificar el código ni volver a compilar el programa.  
   Gracias a este enfoque, las otras partes del sistema (como la gestión de productos, usuarios o proveedores) pueden usar esta misma conexión de forma segura y sin repetir código, ya que todas acceden a la base a través del mismo punto.
6. Unit of Work: Este patrón se usó para asegurar que todas las operaciones relacionadas con una venta (crear la venta, descontar productos del inventario, registrar el detalle, etc.) se ejecuten como una única transacción. Si ocurre un error, se anulan todas las operaciones para evitar inconsistencias en la base de datos.
7. Factory: Se utilizó un patrón Factory para la creación de objetos complejos como las ventas y los reportes. Por ejemplo, al crear una nueva factura, el sistema genera automáticamente los objetos necesarios para representar el total de la venta, los productos involucrados y el usuario que la realizó, centralizando esa lógica en una clase encargada de ensamblar la estructura completa de la venta.