Enunciado de proyecto:

Desarrollar una aplicación de escritorio en C# que permita gestionar la compra y venta de productos, integrando los conceptos de facturación, inventario y contabilidad, utilizando las siguientes tablas:

1. **com_cabecera**: Gestiona la cabecera de las órdenes de compra, permitiendo registrar la entidad, el subtotal, IVA, total y la relación con un asiento contable.
2. **com_detalle**: Representa el detalle de los productos comprados, incluyendo la cantidad, el precio unitario y el subtotal por producto.
3. **com_lista_precios**: Administra la lista de precios de los productos por entidad, lo que permitirá definir los precios de compra.
4. **con_asiento_cabecera** y **con_asiento_detalle**: Manejan los asientos contables generados automáticamente a partir de las transacciones de compra y venta.
5. **fac_cabecera** y **fac_detalle**: Gestionan la emisión de facturas, incluyendo detalles como los productos vendidos, precios, IVA y la generación del asiento contable relacionado.
6. **inv_item**, **inv_bodega**, **inv_movimiento**: Permiten gestionar el inventario de productos, su movimiento entre bodegas y el control del stock.
7. Entre otras (Ver modelo entidad relación).

Requerimientos:

- Implementar clases que representen las entidades involucradas (Compra, Factura, Producto, Bodega, Movimiento de Inventario, Asiento Contable).
- Utilizar el principio de encapsulamiento para gestionar los atributos de las clases.
- Aplicar herencia en las clases relacionadas con entidades de tipo común, como las transacciones de compra y venta.
- Implementar la lógica para registrar y consultar transacciones de compra, ventas, movimientos de inventario y asientos contables.
- El sistema deberá validar la correcta asociación entre las transacciones de compra/venta y los asientos contables generados.

Extras:

- Implementar una interfaz gráfica básica utilizando Windows Forms o WPF.
- Almacenar los datos en una base de datos SQL Server y permitir consultas y reportes básicos desde la interfaz de usuario.
- Utilizar una arquitectura por capas.

Capas del proyecto

1. **Capa de Presentación (UI - Interfaz de Usuario):**
    - Es responsable de la interacción con el usuario, a través de la cual se ingresan los datos y se muestran los resultados.
    - En esta capa se pueden utilizar tecnologías como **Windows Forms** o **WPF**.
    - La capa de presentación se comunica únicamente con la **Capa de Negocio**.
2. **Capa de Negocio (BL - Lógica de Negocio):**
    - Aquí se encuentran las reglas de negocio. Esta capa define cómo funcionan las transacciones y operaciones que ocurren en la aplicación (compras, ventas, generación de asientos contables).
    - Esta capa actúa como un intermediario entre la capa de presentación y la capa de acceso a datos.
    - El principal propósito de esta capa es mantener las reglas de negocio separadas de la interfaz de usuario y la base de datos.
3. **Capa de Acceso a Datos (DAL - Data Access Layer):**
    - Esta capa maneja todas las interacciones con la base de datos, como consultas, inserciones, actualizaciones y eliminaciones de información.
    - Aquí se implementan métodos que permiten acceder y manipular los datos almacenados en SQL Server.
    - Se utilizan tecnologías como **Entity Framework** o **ADO.NET** para realizar la conexión a la base de datos.

4. **Capa de Entidades o Modelos (Entities/DTO - Data Transfer Objects):**
    - Contiene las clases que representan los objetos del dominio de la aplicación, como Producto, Factura, Asiento Contable, entre otros.
    - Sirve como una representación de las tablas de la base de datos en código y permite transportar los datos entre las capas de la aplicación.
5. **Capa de Servicios (Opcional - Servicios REST o API):**
    - Si se requiere escalabilidad o se tiene la intención de exponer algunos servicios a otros sistemas o aplicaciones, se puede incluir una capa de servicios.
    - Aquí se implementan API REST o servicios web que se comunican con la capa de negocio y exponen funcionalidades a otras aplicaciones.

Sugerencia de arquitectura:

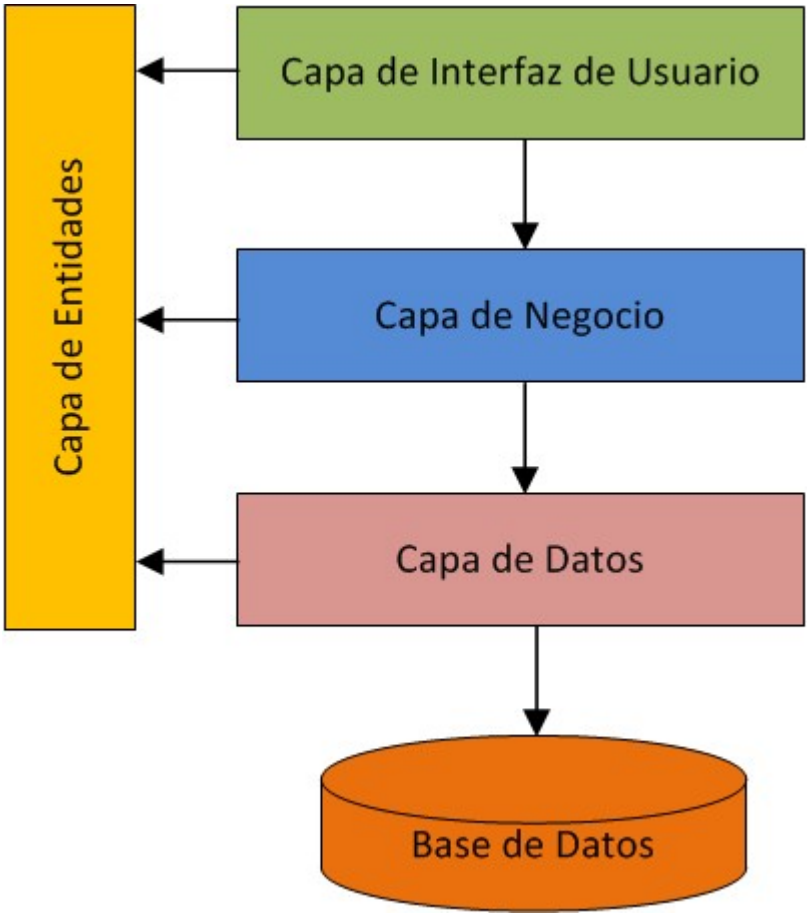
La **arquitectura por capas** es ideal para mantener el código estructurado y fácilmente mantenible. Además, permite hacer cambios en una capa sin afectar a las demás. En este caso, se propone:

1. **Capa de Presentación** (Windows Forms/WPF)
2. **Capa de Negocio** (Manejo de lógica de inventarios, compras, ventas y contabilidad)
3. **Capa de Acceso a Datos** (ADO.NET o Entity Framework)
4. **Capa de Entidades** (Modelos de dominio)

Arquitectura sugerida:

La **arquitectura en capas tradicional** es la mejor para este proyecto debido a su simplicidad y separación de responsabilidades. Además, es escalable y flexible, lo que permite agregar nuevas funcionalidades sin complicaciones, y facilita las pruebas unitarias de cada componente.

Gráfico de la arquitectura a utilizar



https://saov.wordpress.com/2013/04/16/programacion-en-n-capas/

Información adicional

La **Capa de Acceso a Datos (Data Access Layer - DAL)** es la encargada de interactuar con la base de datos y realizar operaciones como inserciones, actualizaciones, eliminaciones o consultas. En C#, se puede implementar la DAL utilizando tecnologías como **ADO.NET** o **Entity Framework (EF)**, y cada una tiene diferencias significativas en cómo se conecta y maneja los datos.

Diferencias entre ADO.NET y Entity Framework

1. Nivel de Abstracción

- **ADO.NET**: Es una tecnología más directa y de bajo nivel para interactuar con la base de datos. Proporciona un control más fino sobre las consultas SQL y la manipulación de datos. El programador escribe el código SQL manualmente.
- **Entity Framework**: Es un **ORM (Object-Relational Mapping)** que genera automáticamente el SQL a partir de las entidades definidas en el código.

Ofrece un nivel de abstracción más alto, lo que facilita la interacción con la base de datos sin necesidad de escribir consultas SQL directamente.

2. Manejo de las Consultas

- **ADO.NET**: Las consultas SQL son escritas manualmente dentro del código (ej. SELECT, INSERT, UPDATE, DELETE). Se usa SqlCommand para ejecutar estas consultas.
- **Entity Framework**: Las consultas SQL se generan automáticamente a partir de consultas en **LINQ** o expresiones lambda, trabajando directamente con entidades (clases que representan tablas de la base de datos).

3. Desempeño

- **ADO.NET**: Generalmente es más rápido, ya que trabaja directamente con consultas SQL escritas a mano, eliminando la sobrecarga de traducir código a consultas SQL.
- **Entity Framework**: Puede ser más lento en ciertas situaciones debido a la capa adicional que traduce las consultas LINQ a SQL. Sin embargo, es más fácil de usar y de mantener.

4. Complejidad

- **ADO.NET**: Más complejo y verboso, ya que requiere manejar manualmente las conexiones, comandos y parámetros. Ideal cuando se necesita un control total sobre las consultas y la interacción con la base de datos.
- **Entity Framework**: Más sencillo y fácil de usar para la mayoría de las operaciones CRUD (Create, Read, Update, Delete), permitiendo trabajar directamente con objetos del dominio sin preocuparse por el SQL subyacente.

5. Mantenimiento

- **ADO.NET**: Si se cambia la estructura de la base de datos, el programador debe ajustar el código manualmente.
- **Entity Framework**: Si se usa el enfoque **Code First** o **Database First**, EF ajusta automáticamente las clases o genera código en función de los cambios en la base de datos.

Diagrama entidad relación