El objetivo de esta aplicación es crear un sistema distribuido de gestión de almacenes para una empresa que tiene varios almacenes separados. Cada almacén tendrá su propio nodo (un servidor con su base de datos local) y el sistema permitirá gestionar la información de los productos, el stock y la sincronización de datos entre los diferentes almacenes.

Requisitos Funcionales

1. **Gestión de Productos**:
    a. Registrar nuevos productos en el inventario de cada almacén.
    b. Modificar información de productos (nombre, descripción, precio, cantidad).
    c. Eliminar productos.
    d. Reservar productos y eliminarlos de reservados
    e. Consultar el inventario disponible en cada almacén.
2. **Gestión de Movimientos de Mercancías**:
    a. Registrar entradas y salidas de productos (ventas, compras, traslados).
    b. Generar reportes de movimientos de mercancías (historial de entradas y salidas).
3. **Comunicación entre Servidores**:
    a. Los servidores pueden intercambiar información sobre el inventario y movimientos de mercancías mediante solicitudes REST.
4. **Gestión de Base de Datos Local**:
    a. Cada almacén tiene su propia base de datos SQLite.
    b. Las consultas a la base de datos local se gestionan secuencialmente mediante la cola de mensajes (ZMQ).
5. **Interfaz Cliente (Consola)**:
    a. Un cliente de texto simple que permite realizar acciones de consulta y gestión de inventarios de forma básica.

Requisitos No Funcionales

1. **Escalabilidad**:
    - El sistema debe ser escalable, permitiendo agregar más servidores (almacenes) a medida que la empresa crece.
    - Los servidores deben poder desconectarse y reconectarse de forma dinámica.
2. **Rendimiento**:
    - El servidor web debe ser capaz de manejar múltiples peticiones simultáneas de clientes y servidores sin afectar su rendimiento.
    - La comunicación con la base de datos debe ser eficiente, utilizando la cola de mensajes para coordinar las consultas.
3. **Seguridad**:
    - Los clientes y servidores deben comunicarse de manera segura utilizando HTTPS.
    - Los datos sensibles deben ser encriptados y protegidos adecuadamente.
4. **Fiabilidad**:
    - El sistema debe ser tolerante a fallos, permitiendo reconectar servidores y clientes en caso de desconexión.

Tecnologías Utilizadas

1. **Lenguaje de Programación**: Python (para el servidor web REST y el cliente de texto).
2. **Servidor Web REST**: Flask o FastAPI.
3. **Base de Datos**: SQLite.
4. **Cola de Mensajes (MQ)**: ZMQ (para coordinar las peticiones y respuestas entre el servidor web y el manejador de base de datos).
5. **ORM**: SQLAlchemy (para interactuar con SQLite).
6. **Cliente de Texto**: Cliente de consola básico utilizando Python.