Carlos Alberto Barrera Lugo

cbarreralugo@gmail.com

Resumen

Requiere un desarrollo en .Net Core y SQL Server que ayude a gestionar el inventario en sus sucursales y llevar un registro de las ventas y compras de productos.

PRoyecto acme

Prueba de conocimientos .net

Contenido

[Problema 2](#_Toc155285741)

[Instrucciones 2](#_Toc155285742)

[Análisis del planteamiento dado. 3](#_Toc155285743)

[Decisión de plataforma a desarrollar. 5](#_Toc155285744)

[Sugerencia para el proyecto ACME: 6](#_Toc155285745)

[1. Planificación: 7](#_Toc155285746)

[2. Diseño de la Base de Datos: 7](#_Toc155285747)

[Tablas Necesarias: 7](#_Toc155285748)

[Diagrama 8](#_Toc155285749)

[3. Desarrollo: 8](#_Toc155285750)

[4. Pruebas: 9](#_Toc155285751)

[5. Despliegue: 9](#_Toc155285752)

[6. Mantenimiento y Actualización: 9](#_Toc155285753)

[Herramientas y Tecnologías Sugeridas: 9](#_Toc155285754)

# Problema

* La empresa marca ACME vende productos varios en todas sus sucursales
* El inventario lo tienen en una hoja de cálculo lo cual complica su actualización en tiempo real.
* Requiere un desarrollo en .Net Core y SQL Server que ayude a gestionar el inventario en sus sucursales y llevar un registro de las ventas y compras de productos.
* Una de las ventajas del sistema es que podría consultar en que sucursal hay existencias de un producto en particular, así como la cantidad en cada una.

# Instrucciones

* Generar una aplicación que resuelva el problema con las siguientes características:
* 1. Cuente con una base de datos normalizada en SQL Server
* 2. Implemente CQRS y DDD
* 3. Considere algún patrón de resiliencia
* 4. Aplicación utilizando contenedor Linux

Mínimo requerido:

Base de datos SQL Server

Actualización de las existencias de un producto (Compra)

Consulta de existencia de algún producto en las diferentes sucursales

# Análisis del planteamiento dado.

**1. Comprender el Problema y Requisitos:**

* **Problema principal**: La gestión de inventario en tiempo real es ineficaz con el actual sistema de hojas de cálculo.
* **Necesidad**: Un sistema centralizado en .Net Core y SQL Server para manejar inventario, ventas y compras en todas las sucursales.

**2. Diseño y Arquitectura:**

**Base de Datos (SQL Server):**

* **Normalización**: Diseñar una base de datos normalizada para evitar redundancias y asegurar la integridad de los datos.
* **Entidades clave**: Productos, Sucursales, Inventario, Compras, Ventas.

**Aplicación .Net Core:**

* **CQRS (Command Query Responsibility Segregation):**
  + **Comandos para operaciones de escritura**: Actualizar inventario, registrar ventas y compras.
  + **Consultas para operaciones de lectura**: Verificar existencias en sucursales.
* **DDD (Domain-Driven Design):**
  + **Modelo de dominio rico**: Encapsular lógica de negocio compleja.
  + **Bounded Contexts**: Separar distintas áreas del negocio para evitar complejidades.
* **Patrón de Resiliencia:**
  + Considerar implementaciones como Circuit Breaker o Retry Pattern para manejar fallos transitorios en la base de datos o llamadas de API.

**Contenedores Linux:**

* **Docker**: Desplegar la aplicación y la base de datos en contenedores para facilitar la distribución y el escalado.

**3. Desarrollo y Pruebas:**

* **Implementar Funcionalidades Mínimas Requeridas:**
  + Actualización de existencias de productos.
  + Consulta de existencia de productos en diferentes sucursales.
* **Pruebas**: Asegurar que la aplicación maneja correctamente la actualización de inventario y las consultas en tiempo real.

**4. Mejores Prácticas:**

* **Código limpio y documentado**: Facilitar el mantenimiento y la escalabilidad.
* **Seguridad**: Asegurar la protección de datos sensibles.
* **Optimización del rendimiento**: Particularmente importante para las operaciones de base de datos y las respuestas en tiempo real.

**5. Despliegue:**

* **Contenedores**: Utilizar Docker para crear un entorno consistente para desarrollo, pruebas y producción.
* **Monitoreo y Mantenimiento**: Establecer procesos para el monitoreo del sistema y la rápida solución de problemas.

**Pasos Siguientes:**

1. **Análisis de Requisitos Detallado**: Trabajar con stakeholders para entender todas las necesidades y prioridades.
2. **Diseño de la Base de Datos y la Aplicación**: Iniciar con la modelación de la base de datos y la arquitectura de la aplicación.
3. **Desarrollo Iterativo**: Implementar características en ciclos cortos, priorizando el mínimo producto viable.
4. **Pruebas Continuas**: Asegurar calidad y rendimiento en cada etapa del desarrollo.
5. **Despliegue y Monitoreo**: Desplegar la aplicación y monitorear su rendimiento y uso.

**Conclusión:**

Este proyecto requiere una combinación cuidadosa de tecnología y patrones de diseño para crear una solución robusta y eficiente.

La utilización de .Net Core, SQL Server, CQRS, DDD, y contenedores Linux son decisiones acertadas que contribuirán al éxito del sistema de gestión de inventario.

Con la planificación adecuada y un enfoque iterativo y centrado en el usuario, se puede construir una solución que no solo resuelva los problemas actuales, sino que también sea escalable para el futuro.

# Decisión de plataforma a desarrollar.

La elección entre una aplicación web, de escritorio, móvil u otra depende de varios factores, incluyendo el entorno de usuario, requisitos de acceso, facilidad de mantenimiento y actualización, y los recursos disponibles. Dado el contexto del proyecto para ACME, aquí hay algunas consideraciones para cada tipo:

**1. Aplicación Web:**

* **Ventajas:**
  + **Acceso Ubicuo**: Accesible desde cualquier lugar con conexión a internet.
  + **Fácil actualización**: Las actualizaciones se implementan en el servidor y no requieren acción del usuario.
  + **Compatibilidad**: Funciona en cualquier dispositivo o sistema operativo con un navegador web.
* **Desventajas:**
  + **Dependencia de Internet**: Requiere conexión constante a internet.
  + **Seguridad**: Exposición a vulnerabilidades web comunes si no se implementan adecuadas medidas de seguridad.

**2. Aplicación de Escritorio:**

* **Ventajas:**
  + **Rendimiento**: Generalmente más rápido y eficiente en recursos del sistema.
  + **Funcionalidad Offline**: Puede funcionar sin conexión a internet.
  + **Seguridad**: Menor exposición a ataques comparado con aplicaciones web.
* **Desventajas:**
  + **Compatibilidad**: Necesita ser desarrollado y mantenido para diferentes sistemas operativos.
  + **Distribución y Actualización**: Más complejo, cada usuario necesita instalar o actualizar el software.

**3. Aplicación Móvil:**

* **Ventajas:**
  + **Portabilidad**: Acceso en cualquier momento y lugar.
  + **Interfaz de Usuario Optimizada**: Diseñada para ser intuitiva y fácil de usar en dispositivos móviles.
* **Desventajas:**
  + **Desarrollo**: Requiere desarrollo para diferentes plataformas (iOS, Android).
  + **Pantalla Pequeña**: Limitaciones debido al tamaño de pantalla y hardware.

## Sugerencia para el proyecto ACME:

Para un sistema de gestión de inventario que requiere actualizaciones en tiempo real y accesibilidad en múltiples ubicaciones, una **aplicación web** es probablemente la opción más adecuada.

Las aplicaciones web son más fáciles de mantener y actualizar, y el personal puede acceder al sistema desde cualquier sucursal o ubicación con acceso a internet.

Además, con el uso de tecnologías modernas de desarrollo web y prácticas de diseño responsivo, la aplicación puede ser accesible y funcional en dispositivos móviles, proporcionando así una buena flexibilidad y experiencia de usuario.

Sin embargo, la elección final debe estar basada en una comprensión profunda de las necesidades específicas del negocio, las expectativas de los usuarios y el entorno operativo de ACME.

Es crucial también considerar los recursos y habilidades disponibles para el desarrollo y mantenimiento a largo plazo de la aplicación seleccionada.

# 1. Planificación:

* **Definir requerimientos**: Listar todas las funcionalidades que se necesitan, como actualización de inventario, visualización de existencias, etc.
* **Esquematizar el flujo de la aplicación**: Definir cómo los usuarios interactuarán con la aplicación y cómo se moverán a través de las distintas funcionalidades.
* **Establecer objetivos y prioridades**: Decidir qué funcionalidades son esenciales para el lanzamiento inicial y cuáles pueden desarrollarse más adelante.

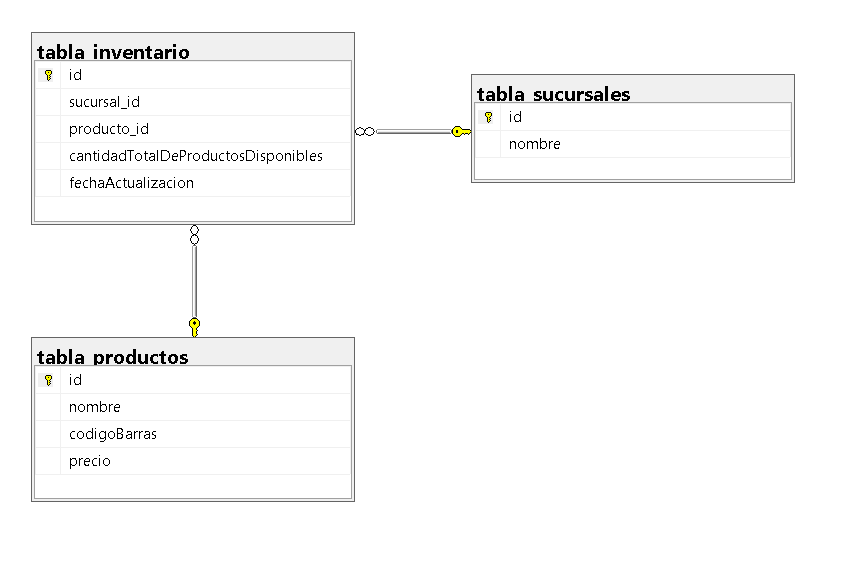
# 2. Diseño de la Base de Datos:

* **Modelar las entidades**: Definir las tablas necesarias como Productos, Sucursales, Inventario, etc.
* **Normalizar la base de datos**: Asegurar de que la base de datos está normalizada para evitar redundancias.
* **Definir relaciones**: Establecer las relaciones entre las tablas, como llaves foráneas, índices, etc.

## Tablas Necesarias:

* + **tabla\_sucursales:**
    - **id** (INT, PK): Identificador único de la sucursal.
    - **nombre** (VARCHAR): Nombre de la sucursal.
  + **tabla\_productos:**
    - **id** (INT, PK): Identificador único del producto.
    - **nombre** (VARCHAR): Nombre del producto.
    - **codigoBarras** (VARCHAR): Código de barras del producto.
    - **precio** (DECIMAL): Precio unitario del producto.
  + **tabla\_inventario:**
    - **id** (INT, PK): Identificador único del inventario.
    - **sucursal\_id** (INT, FK a tabla\_sucursales): Identificador de la sucursal.
    - **producto\_id** (INT, FK a tabla\_productos): Identificador del producto.
    - **cantidadTotalDeProductosDisponibles** (INT): Cantidad total de ese producto disponible en la sucursal.
    - **fechaActualizacion** (DATETIME): Fecha de la última actualización del inventario para ese producto en esa sucursal.

## Diagrama



# 3. Desarrollo:

* **Backend (.Net Core):**
  + **Configurar el entorno de desarrollo**: Instalar .NET Core, SQL Server y cualquier otra herramienta necesaria.
  + **Crear la API**: Desarrollar los endpoints necesarios para las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) y la lógica de negocio.
  + **Implementar CQRS y DDD**: Separar las consultas de las operaciones de actualización y organiza el código siguiendo los principios de diseño orientado al dominio.
  + **Añadir seguridad**: Implementar autenticación y autorización.
* **Frontend (cualquier framework moderno como React, Angular, Vue):**
  + **Diseñar la interfaz de usuario**: Crear mockups y luego desarrollar los componentes de la interfaz.
  + **Conectar con el backend**: Usar llamadas API para interactuar con el backend.
  + **Pruebas**: Realizar pruebas unitarias y de integración.
* **Patrones de Resiliencia**: Implementar patrones como Circuit Breaker para manejar fallos en la base de datos o servicios externos.

# 4. Pruebas:

* **Unitarias**: Para cada componente y servicio.
* **Integración**: Verificar que todos los componentes funcionan juntos como se espera.
* **Usuario**: Realizar pruebas manuales para asegurar que la experiencia del usuario es la esperada.

# 5. Despliegue:

* **Contenedores Docker**: Crear imágenes de Docker para la aplicación y la base de datos.
* **Orquestación**: Considerar usar una herramienta de orquestación como Kubernetes si es necesario.
* **Monitoreo y Logs**: Configurar el monitoreo y los logs para asegurar el rendimiento y facilitar la depuración.

# 6. Mantenimiento y Actualización:

* **Feedback del usuario**: Recoger y analizar el feedback del usuario para mejorar la aplicación.
* **Actualizaciones**: Planear y ejecutar actualizaciones periódicas para añadir nuevas funcionalidades y corregir errores.

# Herramientas y Tecnologías Sugeridas:

* **Backend**: .NET Core
* **Arquitectura**: MVC (Model – View – Controller), usualmente terminamos usando más capas, pero la base es MVC
* **Interacción con Base de Datos**: Entity Framework Core, un ORM popular para .NET
* **Mantenimiento de información de Base de datos**: Stored Procedure
* **Base de datos**: SQL Server
* **Frontend**: Bootstrap, React, Angular o Vue.js
* **Contenedores**: Docker para empaquetar y desplegar la aplicación
* **Versionamiento**: Git para control de versiones
* **Automatización y Despliegue**: GitHub Actions o cualquier otra herramienta de CI/CD