

# Ejercicio 8. Practica 1. Ejercicios sobre Metodologías de Desarrollo en Cascada con RUP.

## 1. Análisis de Requerimientos

**Ejercicio 1: Un cliente te solicita una aplicación web para gestionar su inventario. Define los requisitos funcionales y no funcionales del sistema.**

Requisitos Funcionales:

- a. Requisitos de Negocio:
  - 1. Registro de usuarios con información básica y credenciales de inicio de sesión.
  - 2. Gestión de productos con capacidad para agregar, eliminar y actualizar información de productos.
  - 3. Búsqueda de productos por nombre o categoría.
  - 4. Registro de ordenes de inventario, como compras, ventas y ajustes de stock.
- b. Requisitos de Usuario:
  - 1. Interfaz de usuario intuitiva y fácil de usar.
  - 2. Funcionalidad de categorización de productos para facilitar la organización del inventario.
  - 3. Generación de informes personalizables sobre el estado del inventario y las órdenes.
- c. Requisitos de Sistema:
  - 1. Capacidad para manejar grandes volúmenes de datos de inventario y órdenes.
  - 2. Seguridad de datos mediante cifrado y control de acceso.
  - 3. Integración con sistemas de terceros, como proveedores o sistemas de pago en línea.
  - 4. Mantenimiento de registros de auditoría para rastrear cambios en el inventario y las órdenes.

Requisitos No Funcionales:

- a. Requisitos de Desempeño:
  - 1. Tiempo de respuesta rápido para consultas de inventario y transacciones.
  - 2. Capacidad para escalar para manejar un aumento en la carga de trabajo.
- b. Requisitos de Seguridad:
  - 1. Protección de datos confidenciales de usuarios y transacciones.
  - 2. Autenticación y autorización adecuadas para acceder a funciones sensibles del sistema.
- c. Requisitos de Usabilidad:
  - 1. Interfaz de usuario intuitiva y consistente.
  - 2. Capacidades de personalización para adaptarse a diferentes preferencias de usuario.
- d. Requisitos de Mantenimiento:
  - 1. Código bien documentado y modular para facilitar futuras actualizaciones.
  - 2. Procedimientos de respaldo y restauración para garantizar la disponibilidad y la integridad de los datos.
- e. Requisitos de Fiabilidad:
  - 1. Alta disponibilidad del sistema para minimizar el tiempo de inactividad no planificado.
  - 2. Tolerancia a fallos para garantizar la continuidad del servicio en caso de errores o interrupciones.
- a. Requisitos de Portabilidad:
  - 1. Compatibilidad con diferentes navegadores web y dispositivos.
  - 2. Facilidad de despliegue en diferentes entornos, como servidores locales o en la nube.

**Ejercicio 2: Redacta un caso de uso para la funcionalidad de "Agregar un nuevo producto" en la aplicación web del ejercicio 1.**

Caso de Uso para "Agregar un nuevo producto":

Actor: Usuario Administrador, Usuario empleado.

Descripción: El usuario ingresa los detalles del nuevo producto al sistema.

Precondiciones:

1. El usuario inicia sesión en la aplicación.
2. El usuario accede a la sección de “Gestión de inventario”.

Flujo Principal:

1. El usuario selecciona la opción de “Agregar un nuevo producto”.
2. El sistema muestra un formulario para ingresar los detalles del producto.
3. El usuario ingresa la información del producto (nombre, descripción, cantidad, etc.).
4. El usuario confirma el nuevo del producto.
5. El sistema valida la información y agrega el producto al inventario.
6. El sistema muestra un mensaje de confirmación al usuario.

Flujo Alternativo:

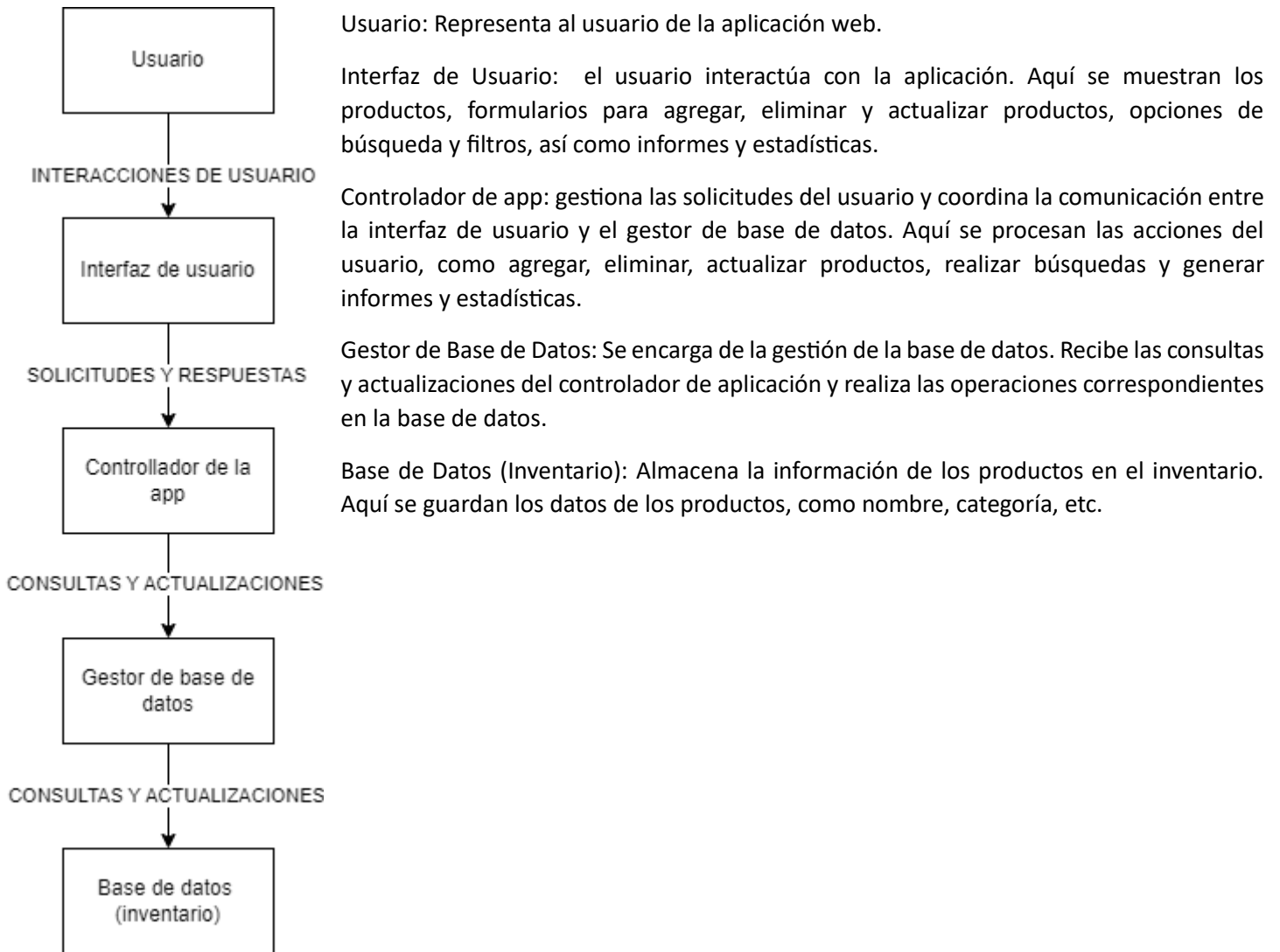
- 5.1. Si los campos obligatorios no son completados o no son válidos, el sistema muestra mensaje de error.
- 5.2. El usuario ingresa los datos correctos en los campos correspondientes e intenta nuevamente ingresar el producto al inventario.

Postcondiciones:

1. El producto se muestra al usuario en la sección de “Gestión de inventario”.

## **2. Diseño del Sistema**

**Ejercicio 3: Elabora un diagrama de flujo de datos para la aplicación web del ejercicio 1.**



**Ejercicio 4: Diseña la interfaz de usuario para la pantalla de "Inicio" de la aplicación web del ejercicio 1.**

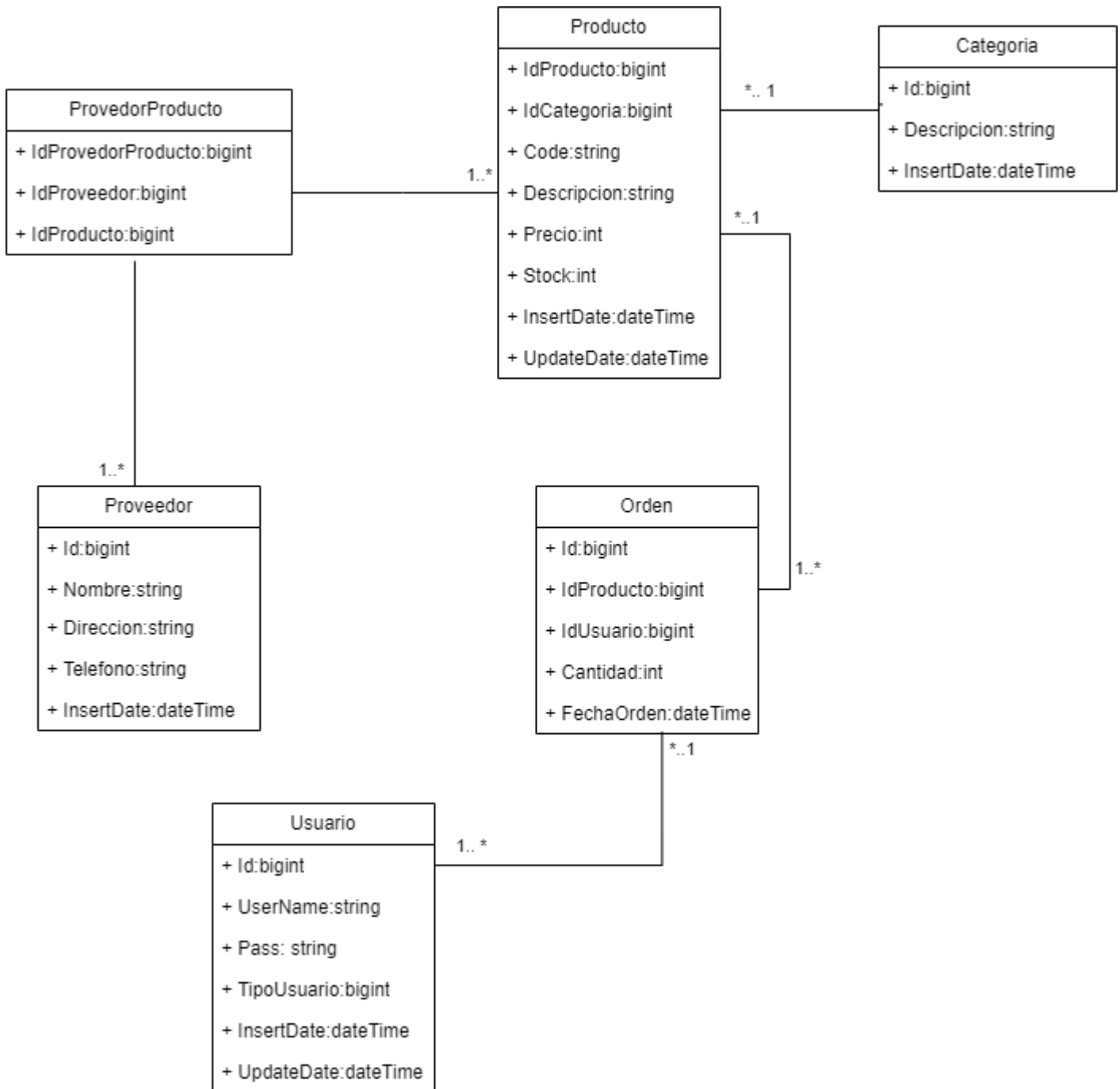
<https://www.figma.com/design/LQfOxs3XvJJB0GnHQCyfHU/GestionInventario?m=dev&node-id=0%3A1&t=gsnAUIOgfuzADSp8-1>

### **3. Diseño del Programa**

**Ejercicio 5: Elige una arquitectura adecuada para la aplicación web del ejercicio 1 y justifica tu elección.**

Se elige la arquitectura MVC debido a su capacidad para separar claramente la lógica de negocio (Modelo), la presentación de datos (Vista) y la lógica de control (Controlador). Esto facilita la escalabilidad, el mantenimiento y la colaboración entre desarrolladores.

**Ejercicio 6: Diseña la base de datos para la aplicación web del ejercicio 1.**



#### 4. Diseño

Utilizando los siguientes diagramas resuelva los casos de usos de los ejercicios 7 y 8:

- Diagrama de Dominio: Identifica las entidades, atributos y relaciones del sistema.
- Diagrama de Robustez: Analiza cómo el sistema responde a diferentes escenarios de uso.
- Prototipo: Crea una versión simplificada del sistema para probar la usabilidad y funcionalidad.
- Diagrama de Secuencia: Describe la interacción entre los diferentes objetos del sistema.
- Diagrama de Clases: Define las clases, sus atributos, métodos y relaciones

**Ejercicio 7: Implementa la funcionalidad de "Agregar un nuevo producto" en la aplicación web del ejercicio 1 utilizando el lenguaje de programación de tu preferencia.**

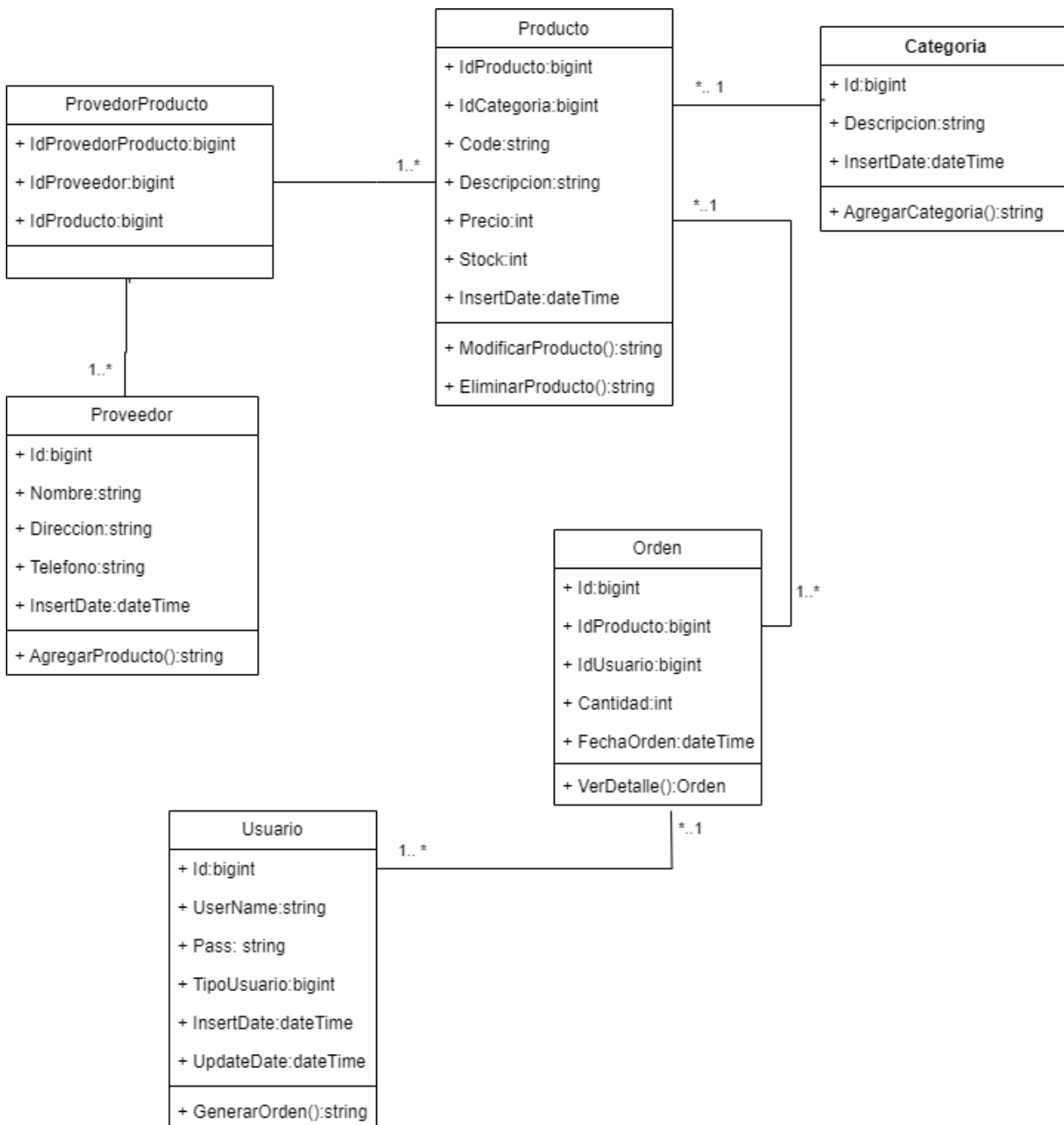
## Diagrama de dominio.

### Entidades:

- Producto. Atributos: Id, Code, Descripcion, Precio, Stock, InsertDate, UpdateDate.
- Categoria. Atributos: Id, Descripcion, InsertDate.
- Proveedor. Atributos: Id, Nombre, Direccion, Telefono, InsertDate.
- Usuario. Atributos: Id, UserName, Pass, TipoUsuario, InsertDate, UpdateDate.
- Orden. Atributos: Id, IdProducto, IdUsuario, Cantidad, FechaOrden.

### Relaciones:

- Un Proveedor puede agregar muchos productos.
- Varios productos pueden tener una misma categoría.
- Una orden puede tener varios productos.
- Un usuario puede tener muchas órdenes.



Ejercicio 8: Implementa la lógica de negocio para la funcionalidad de "Agregar un nuevo producto" en la aplicación web del ejercicio 1.

```
using System;
using System.Linq;
public class Producto
{
    public int Id { get; set; }
    public string Nombre { get; set; }
    public string Descripcion { get; set; }
    public decimal Precio { get; set; }
    public int Stock { get; set; }
}
public class ProductoManager
{
    public bool AgregarProducto(string nombre, string descripcion, decimal precio, int stock)
    {
        using (var context = new DbContext())
        {
            if (string.IsNullOrEmpty(nombre) || precio <= 0 || stock < 0)
            {
                return "Error. El nombre puede estar vacío. Recuerde que el precio y stock debe ser mayor a 0.";
            }
            if (!context.Producto.Any(x => x.Nombre == nombre))
            {
                return "Ya existe un producto con ese nombre.";
            }
            Producto productoNuevo = new Producto
            {
                Nombre = nombre,
                Descripcion = descripcion,
                Precio = precio,
                Stock = stock,
                InsertDate = DateTime.Now
            };
            context.Add(productoNuevo);
            context.SaveChanges();
            return "Producto agregado correctamente.";
        }
    }
}
class Program
{
    static void Main(string[] args)
    {
        ProductoManager manager = new ProductoManager();
        string resultado = manager.AgregarProducto("Nuevo Producto", "Descripción del nuevo producto", 10.99m, 100);
        Console.WriteLine(resultado);
    }
}
```

```
}
```

## 5. Pruebas:

**Ejercicio 9: Define un conjunto de pruebas unitarias para la funcionalidad de "Agregar un nuevo producto" en la aplicación web del ejercicio 1.**

**Prueba de Datos Válidos:** Esta prueba verifica que el método `AgregarProducto` maneje correctamente los datos válidos del producto, es decir, cuando se proporcionan todos los campos requeridos y tienen valores válidos. Para esta prueba, se debe proporcionar un nombre de producto, una descripción, un precio válido (mayor que cero) y un stock válido (no negativo). Devuelve un mensaje de correcto, en caso de ser correcto, de caso contrario reporta el error.

**Prueba de Producto Duplicado:** Verifica que el método `AgregarProducto` maneje correctamente el caso en el que se intenta agregar un producto que ya existe en la base de datos. Se proporcionan datos de un producto que ya existe en la base de datos. Se espera que el método retorne un mensaje de error indicando que el producto ya existe.

**Prueba de Precio Mínimo:** Verifica que el método `AgregarProducto` maneje correctamente el caso en el que se proporciona el precio mínimo para el producto. Se proporciona un nombre de producto, una descripción, el precio mínimo admitido (p. ej., 0.01) y un stock válido. Se espera que el método retorne un mensaje indicando que el producto fue agregado correctamente, de caso contrario, reporta el error.

**Ejercicio 10: Ejecuta pruebas de integración para la funcionalidad de "Agregar un nuevo producto" en la aplicación web del ejercicio 1.**

```
using NUnit.Framework;
using System.Linq;
[TestFixture]
public class ProductoManagerTests
{
    [Test]
    public void AgregarProductoYVerificarBaseDeDatos()
    {
        // Arrange
        ProductoManager manager = new ProductoManager();
        string nombreProducto = "Nuevo Producto";
        string descripcionProducto = "Descripción del nuevo producto";
        decimal precioProducto = 10.99m;
        int stockProducto = 100;
        // Act
        string resultado = manager.AgregarProducto(nombreProducto, descripcionProducto,
precioProducto, stockProducto);
        // Assert
        Assert.AreEqual("Producto agregado correctamente.", resultado);
        using (var context = new DbContext())
        {
            var productoAgregado = context.Productos.FirstOrDefault(p => p.Nombre ==
nombreProducto);
            Assert.IsNotNull(productoAgregado);
            Assert.AreEqual(descripcionProducto, productoAgregado.Descripcion);
            Assert.AreEqual(precioProducto, productoAgregado.Precio);
            Assert.AreEqual(stockProducto, productoAgregado.Stock);
        }
    }
}
```

## **6. Despliegue del Programa:**

### **Ejercicio 11: Definir un plan de despliegue para la aplicación web del ejercicio 1.**

El plan de despliegue para la aplicación web de gestión de inventario consistirá en los siguientes pasos:

1. Configuración del servidor de producción: Adquirir un servidor adecuado para alojar la aplicación web. Configurar el sistema operativo y los servicios necesarios, como el servidor web (por ejemplo, Apache o Nginx) y el motor de base de datos (por ejemplo, SQLServer).
2. Configuración de la base de datos: Crear una base de datos para la aplicación. Configurar las tablas y relaciones según el diseño de la base de datos.
3. Implementación de la aplicación web: Transferir los archivos de la aplicación al servidor de producción. Configurar la aplicación para que funcione correctamente en el entorno de producción.
4. Realización de pruebas de aceptación: Verificar que la aplicación funcione correctamente en el servidor de producción. Realizar pruebas exhaustivas para garantizar que todas las funcionalidades estén operativas y que no haya problemas de rendimiento o errores.

### **Ejercicio 12: Despliega la aplicación web del ejercicio 1 en un servidor de producción.**

1. Conexión al servidor de producción: Acceder al servidor a través de SSH u otro método seguro.
2. Clonado del repositorio de la aplicación: Clonar el repositorio de la aplicación desde el sistema de control de versiones (por ejemplo, Git).
3. Configuración del entorno de producción: Configurar las variables de entorno y ajustes de configuración específicos para el entorno de producción.
4. Instalación de dependencias: Instalar todas las dependencias de la aplicación utilizando un gestor de paquetes como NuGet o npm.
5. Configuración de la base de datos: Configurar la conexión a la base de datos de producción y ejecutar las migraciones necesarias para asegurar la consistencia del esquema.
6. Ejecución de la aplicación en el servidor: Iniciar el servidor de la aplicación y asegurarse de que esté en funcionamiento correctamente.

## **7. Mantenimiento:**

### **Ejercicio 13: Definir un plan de mantenimiento para la aplicación web del ejercicio 1.**

1. Actualizaciones de seguridad periódicas: Mantener actualizados todos los componentes de software, incluyendo el sistema operativo, el servidor web, el motor de base de datos y cualquier biblioteca o framework utilizado en la aplicación.
2. Monitoreo del rendimiento del servidor: Supervisar el rendimiento del servidor de producción para identificar posibles cuellos de botella o problemas de rendimiento que puedan afectar la experiencia del usuario.
3. Corrección de errores reportados por los usuarios: Establecer un proceso para recibir y priorizar informes de errores por parte de los usuarios, y corregirlos de manera oportuna.
4. Respaldo regular de datos: Realizar copias de seguridad regulares de la base de datos y los archivos de la aplicación para garantizar la disponibilidad de los datos en caso de pérdida o corrupción.
5. Escalado según sea necesario: Monitorizar el crecimiento de la aplicación y escalar los recursos del servidor según sea necesario para manejar la carga adicional.

### **Ejercicio 14: Implementa una corrección de errores para un problema detectado en la aplicación web del ejercicio 1.**

1. **Identificación del Problema:** Identificar el error o problema detectado en la aplicación.
2. **Registro del Problema:** Registrar el problema en el sistema de seguimiento de problemas o en la herramienta de gestión de cambios del proyecto.
3. **Priorización del Problema:** Evaluar la gravedad y el impacto del problema para determinar su prioridad.



4. **Análisis del Problema:** Realizar un análisis detallado para comprender la causa raíz del problema.
5. **Diseño de la Solución:** Diseñar una solución adecuada para abordar el problema identificado.
6. **Implementación de la Corrección:** Implementar la solución propuesta en el código fuente de la aplicación.
7. **Pruebas Unitarias:** Realizar pruebas unitarias para verificar que la corrección haya sido exitosa.
8. **Pruebas de Integración:** Realizar pruebas de integración para asegurar que la corrección no haya introducido nuevos problemas.
9. **Validación del Usuario:** Solicitar la validación de usuarios finales o representantes del cliente para confirmar la corrección.
10. **Despliegue:** Desplegar la corrección en el entorno de producción de la aplicación.
11. **Documentación:** Actualizar la documentación del proyecto para reflejar la corrección implementada.
12. **Seguimiento y Monitoreo:** Seguir de cerca el comportamiento de la aplicación después de implementar la corrección para asegurarse de que el problema no vuelva a surgir y para identificar cualquier efecto secundario no anticipado.

## **8. Nos preparamos para nuevos retos**

**Ejercicio 15: Arme un equipo de trabajo y defina los roles para realizar los ejercicios anteriores para un futuro dominio de aplicación relacionado con inteligencia artificial generativa.**

1. Gerente de Proyecto: Responsable de liderar el equipo y gestionar el proyecto en su conjunto. Coordina la planificación, ejecución y seguimiento del proyecto, asegurando que se cumplan los objetivos y entregables en los plazos establecidos.
2. Analista de Negocio: Responsable de recopilar los requerimientos del cliente y comprender sus necesidades empresariales. Realiza el análisis de negocio para identificar los procesos clave que la aplicación debe soportar, así como los requisitos funcionales y no funcionales asociados.
3. Arquitecto de Software: Responsable de diseñar la arquitectura de la aplicación, definiendo las diferentes capas y componentes que la componen. Define la estructura técnica del sistema y las tecnologías que se utilizarán para su desarrollo.
4. Desarrollador Frontend: Responsable de implementar la interfaz de usuario de la aplicación, utilizando las tecnologías frontend adecuadas. Transforma los diseños proporcionados por el diseñador UX/UI en páginas web interactivas y funcionales.
5. Desarrollador Backend: Responsable de implementar la lógica de negocio y la capa de datos de la aplicación. Diseña y desarrolla los componentes que permiten la gestión de productos, validaciones, persistencia de datos, entre otros.
6. Tester (QA): Responsable de diseñar y ejecutar pruebas para validar la calidad y funcionalidad de la aplicación. Realiza pruebas unitarias, de integración y de sistema para identificar y corregir errores antes de la entrega al cliente.
7. Diseñador UX/UI: Responsable de diseñar la experiencia de usuario y la interfaz gráfica de la aplicación. Crea wireframes, prototipos y diseños visuales que cumplan con los requisitos de usabilidad y estética del cliente.
8. Administrador de Configuración: Responsable de gestionar la configuración y control de versiones del software. Establece y mantiene los procesos y herramientas necesarios para gestionar los cambios en el código y la documentación del proyecto.