**LABORATORIO DE SOFTWARE**

**Trabajo Práctico Integrador**



**MARTIN ALEJANDRO LAMAS**

**DNI: 30.492.150**

**Plataforma utilizada:**

**Microsoft Visual Studio**

Contenido

[Arquitectura software con DDD y múltiples capas 3](#_Toc169296678)

[Capa de Dominio (Domain Layer) 3](#_Toc169296679)

[Capa de Aplicaciones (Application Layer) 3](#_Toc169296680)

[Capa de Infraestructura de Datos (Infrastructure Layer) 4](#_Toc169296681)

[Capa de Presentación (Presentation Layer) 4](#_Toc169296682)

[Inyección de dependencia para desacoplar servicios 5](#_Toc169296683)

[Capa de Aplicación 5](#_Toc169296684)

[Capa de Infraestructura de Datos 5](#_Toc169296685)

[Pruebas Unitarias en un componente de las capas 5](#_Toc169296686)

[ProductoTests 5](#_Toc169296687)

[EstadoProductoTests 6](#_Toc169296688)

[CRUD completo de la entidad producto 8](#_Toc169296689)

[Crear Producto 8](#_Toc169296690)

[Actualizar Producto 9](#_Toc169296691)

[Leer Producto 9](#_Toc169296692)

[Listar Productos 9](#_Toc169296693)

[Eliminar Producto 10](#_Toc169296694)

[Detalles adicionales 10](#_Toc169296695)

[Atributo Estado en la entidad Producto 10](#_Toc169296696)

[Historia de usuario con criterios de aceptación 11](#_Toc169296697)

[Título: Crear Producto 11](#_Toc169296698)

[Criterios de Aceptación 12](#_Toc169296699)

# Arquitectura software con DDD y múltiples capas

En el desarrollo de la arquitectura del software se aplican los conceptos de DDD (Domain Driven Desing) y, contiene una capa de dominio, una capa de infraestructura de datos, una capa de aplicaciones y una capa de presentación básica (solo para poder comprobar los resultados de las operaciones) la cual es por consola.

## Capa de Dominio (Domain Layer)

**Objetivo**: Contener las entidades del dominio, value objects y la lógica de negocio.

**Requisitos**:

* Entidades y Value Objects que representen el núcleo del negocio.
* Lógica de negocio encapsulada dentro de estas entidades y objetos de valor.

Contiene las entidades y los objetos de valor.

**Ejemplo:** La clase Producto y sus objetos de valor como Identificador, Nombre, Descripcion, Precio, CategoriaId, CantidadStock.

## Capa de Aplicaciones (Application Layer)

**Objetivo**: Contener la lógica de aplicación que orquesta la lógica de dominio.

**Requisitos**:

* Casos de uso que gestionen la interacción con la capa de dominio y los repositorios.
* Servicios de aplicación que implementen los casos de uso.

Contiene los casos de uso que orquestan las operaciones de dominio.

**Ejemplo:** Clases como CrearProducto, ListarProducto, ActualizarProducto, EliminarProducto que contienen la lógica para gestionar las operaciones con los productos.

## Capa de Infraestructura de Datos (Infrastructure Layer)

**Objetivo**: Contener las implementaciones de los repositorios y el acceso a los datos.

**Requisitos**:

* Implementaciones de los repositorios que interactúan con la base de datos o almacenamiento en memoria.
* Asegurar que los datos se manejan de manera adecuada

Implementa los repositorios para la persistencia de datos.

**Ejemplo:** La clase ProductoRepositorioEnMemoria que implementa la interfaz ProductoRepositorio para gestionar los productos en una lista en memoria.

## Capa de Presentación (Presentation Layer)

**Objetivo**: Proporcionar una interfaz de usuario para interactuar con la aplicación.

**Requisitos**:

* Una interfaz básica para permitir la interacción con las operaciones CRUD.
* Puede ser una aplicación de consola para pruebas.

Proporciona una interfaz de usuario para interactuar con la aplicación.

**Ejemplo:** Un menú de consola en Program.cs que permite al usuario crear, listar, actualizar y eliminar productos.

# Inyección de dependencia para desacoplar servicios

Se utiliza la inyección de dependencias para desacoplar las clases de casos de uso de las implementaciones concretas de los repositorios, lo que facilita el modularidad y las pruebas del código.

## Capa de Aplicación

En las clases de casos de uso (CrearProducto, ListarProductos, ActualizarProducto, EliminarProducto), se utiliza la inyección de dependencias en los constructores para recibir las implementaciones concretas de los repositorios.

Ejemplo: CrearProducto

## Capa de Infraestructura de Datos

Se definió las implementaciones concretas de los repositorios, como ProductoRepositorioEnMemoria, que implementa las interfaces definidas en la capa de Dominio (ProductoRepositorio).

Ejemplo: ProductoRepositorioEnMemoria

# Pruebas Unitarias en un componente de las capas

## ProductoTests

Prueba: producto\_GettersAndSetters\_WorkCorrectly

**Propósito:** Esta prueba verifica que los getters y setters de la clase Producto funcionan correctamente.

**Explicación:**

* **Arrange:** Se crea una instancia de Producto con valores iniciales.
* **Act:** Se actualizan los valores del producto usando los setters.
* **Assert:** Se verifica que los getters devuelvan los valores correctos que fueron establecidos previamente.

## EstadoProductoTests

**Prueba:** Constructor\_CadenaValida\_AsignaEstadoCorrectamente

**Propósito:** Verifica que el constructor que recibe una cadena asigne correctamente el estado.

**Explicación:**

* **Arrange:** Se define un estado válido en formato de cadena.
* **Act:** Se crea una instancia de EstadoProducto usando el constructor que recibe una cadena.
* **Assert:** Se verifica que el valor del estado sea el esperado.

**Prueba:** Constructor\_EnteroValido\_AsignaEstadoCorrectamente

**Propósito:** Verifica que el constructor que recibe un entero asigne correctamente el estado.

**Explicación:**

* **Arrange:** Se define un estado válido en formato entero.
* **Act:** Se crea una instancia de EstadoProducto usando el constructor que recibe un entero.
* **Assert:** Se verifica que el valor del estado sea el esperado.

**Prueba:** Constructor\_CadenaInvalida\_LanzaExcepcion

**Propósito:** Verifica que se lance una excepción al intentar crear un EstadoProducto con una cadena inválida.

**Explicación:**

* **Arrange:** Se define un estado inválido en formato de cadena.
* **Act & Assert:** Se verifica que el constructor lance una excepción al recibir una cadena inválida.

**Prueba:** Constructor\_EnteroInvalido\_LanzaExcepcion

**Propósito:** Verifica que se lance una excepción al intentar crear un EstadoProducto con un entero inválido.

**Explicación:**

* **Arrange:** Se define un estado inválido en formato entero.
* **Act & Assert:** Se verifica que el constructor lance una excepción al recibir un entero inválido.

**Prueba:** Valor\_RetornaValorCorrecto

**Propósito:** Verifica que el método Valor devuelva el valor correcto del estado.

**Explicación:**

* **Arrange:** Se crea una instancia de EstadoProducto con un estado específico.
* **Act:** Se obtiene el valor del estado.
* **Assert:** Se verifica que el valor devuelto sea el correcto.

**Prueba:** ToString\_RetornaValorCorrecto

**Propósito:** Verifica que el método ToString devuelva el valor correcto del estado.

**Explicación:**

* **Arrange:** Se crea una instancia de EstadoProducto con un estado específico.
* **Act:** Se llama al método ToString.
* **Assert:** Se verifica que el resultado sea el valor correcto del estado.

**Prueba:** Equals\_CompararConMismoValor\_RetornaVerdadero

**Propósito:** Verifica que dos instancias de EstadoProducto con el mismo valor se consideren iguales.

**Explicación:**

* **Arrange:** Se crean dos instancias de EstadoProducto con el mismo valor.
* **Act & Assert:** Se verifica que las dos instancias sean consideradas iguales.

**Prueba:** Equals\_CompararConValorDiferente\_RetornaFalso

**Propósito:** Verifica que dos instancias de EstadoProducto con diferentes valores no se consideren iguales.

**Explicación:**

* **Arrange:** Se crean dos instancias de EstadoProducto con diferentes valores.
* **Act & Assert:** Se verifica que las dos instancias no sean consideradas iguales.

**Prueba:** GetHashCode\_MismoValor\_RetornaMismoHashCode

**Propósito:** Verifica que dos instancias de EstadoProducto con el mismo valor tengan el mismo código hash.

**Explicación:**

* **Arrange:** Se crean dos instancias de EstadoProducto con el mismo valor.
* **Act & Assert:** Se verifica que ambos objetos tengan el mismo código hash.

# CRUD completo de la entidad producto

## Crear Producto

**Clase**: CrearProducto

**Propósito**: Permite la creación de un nuevo producto.

**Método principal**: Ejecutar(ProductoDTO productoDTO)

* Verifica si el producto con el ID dado ya existe utilizando repositorio.ExisteProductoConId.
* Si el ID no existe, crea una nueva instancia de Producto con los datos del DTO (productoDTO) y lo guarda en el repositorio con repositorio.Grabar.

## Actualizar Producto

**Clase**: ActualizarProducto

**Propósito**: Permite la actualización de un producto existente.

**Método principal**: Ejecutar(ProductoDTO productoDTO)

* Obtiene el producto existente por ID usando repositorio.ObtenerPorId.
* Si el producto existe, actualiza sus propiedades con los datos del DTO (productoDTO) y guarda los cambios con repositorio.Actualizar.

## Leer Producto

**Clase**: LeerProducto

**Propósito**: Permite la consulta de un producto por su ID.

**Método principal**: Ejecutar(int productoId)

* Obtiene el producto por ID usando repositorio.ObtenerPorId.
* Si el producto existe, devuelve un ProductoDTO con los datos del producto; de lo contrario, devuelve null.

## Listar Productos

**Clase**: ListarProducto

**Propósito**: Permite la consulta de todos los productos.

**Método principal**: Ejecutar()

* Obtiene una lista de todos los productos usando repositorio.Listar.
* Convierte cada producto en un ProductoDTO y los añade a una lista que se retorna.

## Eliminar Producto

**Clase**: EliminarProducto

**Propósito**: Permite la eliminación de un producto por su ID.

**Método principal**: Ejecutar(int productoId)

* Obtiene el producto por ID usando repositorio.ObtenerPorId.
* Si el producto existe, lo elimina del repositorio con repositorio.Eliminar.

## Detalles adicionales

* **Inyección de Dependencias**: Cada clase de CRUD recibe una instancia de ProductoRepositorio a través de su constructor. Esto facilita el desacoplamiento entre las capas de aplicación e infraestructura de datos.
* **DTO (Data Transfer Object)**: ProductoDTO se utiliza para transferir datos entre las capas, evitando que las entidades de dominio se expongan directamente a la capa de aplicación.

# Atributo Estado en la entidad Producto

La entidad Producto tiene un atributo que representa un Estado (Activo, Inactivo, Pendiente)

# Historia de usuario con criterios de aceptación

## Título: Crear Producto

**Descripción:** Como administrador del sistema, necesito poder crear nuevos productos en el sistema para mantener actualizado el catálogo de productos disponibles.

**Pasos:**

1. El administrador accede al sistema e inicia sesión con sus credenciales.
2. El administrador navega a la sección de administración de productos.
3. En la sección de administración de productos, el administrador selecciona la opción de "Crear Nuevo Producto".
4. El sistema presenta un formulario para ingresar los detalles del nuevo producto.
5. El administrador completa el formulario proporcionando la siguiente información del producto:
   * **ID del producto:** número único para identificar el producto.
   * **Nombre del producto:** nombre descriptivo del producto.
   * **Descripción del producto:** breve descripción del producto.
   * **Precio del producto:** precio de venta del producto.
   * **ID de la categoría:** identificador de la categoría a la que pertenece el producto.
   * **Cantidad en stock:** cantidad disponible en inventario del producto.
   * **Estado del producto:** estado actual del producto (activo, inactivo, pendiente, etc.).
6. El administrador envía el formulario de creación del producto.
7. El sistema valida la información proporcionada y crea el nuevo producto en la base de datos.
8. El sistema muestra un mensaje de confirmación indicando que el producto ha sido creado exitosamente.

## Criterios de Aceptación

* El formulario de creación de productos debe validar que todos los campos obligatorios sean ingresados.
* El ID del producto debe ser único en el sistema.
* El sistema debe permitir al administrador crear múltiples productos sin problemas.
* Después de crear un producto, este debe aparecer inmediatamente en el catálogo de productos del sistema.
* El sistema debe ser capaz de manejar correctamente cualquier error durante el proceso de creación del producto y mostrar mensajes de error informativos al usuario.

# Instrucciones para Desplegar la Aplicación en un Entorno de Prueba

**Tabla de Contenidos**

1. Requisitos Previos
2. Instalación
3. Configuración de la Base de Datos
4. Ejecución de la Aplicación
5. Solución de Problemas Comunes

## Requisitos Previos

Antes de comenzar, asegúrate de tener instalados los siguientes componentes:

* **.NET SDK**: Puedes descargarlo desde [aquí](https://dotnet.microsoft.com/download/dotnet).
* **Visual Studio**: Herramienta recomendada para el desarrollo y ejecución de aplicaciones .NET. Puedes descargarlo desde [aquí](https://visualstudio.microsoft.com/).
* **MySQL**: Sistema de gestión de bases de datos. Puedes descargarlo desde [aquí](https://dev.mysql.com/downloads/installer/).

## Instalación

1. **Clona el repositorio del proyecto**:

Copiar código

git clone https://github.com/Martincho83/TP-LAB.git

1. **Navega al directorio del proyecto**:

Copiar código

cd TP-LAB

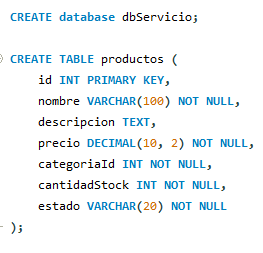
1. **Abre el proyecto en Visual Studio**:
   * Abre Visual Studio.
   * Selecciona "Open a project or solution".
   * Navega al directorio del proyecto y abre el archivo .sln.
2. **Restaura las dependencias del proyecto**:

En la ventana de Visual Studio, selecciona Tools > NuGet Package Manager > Manage NuGet Packages for Solution y luego haz clic en Restore.

## Configuración de la Base de Datos

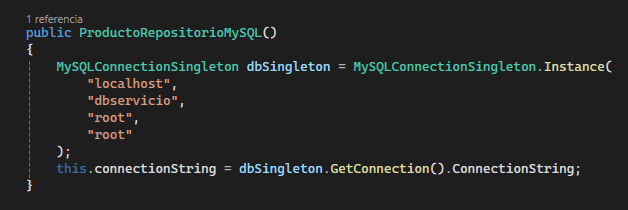
1. **Crear la Base de Datos**:

Ejecuta la siguiente instrucción SQL en tu servidor MySQL para crear la base de datos y la tabla de productos:



1. **Configura la conexión a la base de datos**:

Abre el archivo appsettings.json en el directorio del proyecto y configura la cadena de conexión:



## Ejecución de la Aplicación

1. **Compilar la aplicación**:

En Visual Studio, selecciona Build > Build Solution o presiona Ctrl+Shift+B.

1. **Ejecutar la aplicación**:
   * Selecciona Debug > Start Without Debugging o presiona Ctrl+F5.
   * Esto iniciará la aplicación y estará disponible en la consola.
2. **Acceder a la aplicación**:

Una vez abierta la consola, se puede interactuar con la aplicación y realizar el crud del producto.

**Solución de Problemas Comunes**

* **Error de conexión a la base de datos**:
  + Verifica que MySQL está ejecutándose y que las credenciales en son correctas.
* **Dependencias faltantes**:
  + En Visual Studio, selecciona Tools > NuGet Package Manager > Manage NuGet Packages for Solution y asegúrate de que todas las dependencias estén instaladas.
* **Problemas de compilación**:
  + Asegúrate de estar utilizando la versión correcta del SDK de .NET. Puedes verificar y cambiar la versión del SDK en las propiedades del proyecto en Visual Studio.