.NET MAUI Gestión de Artículos

Status	Not started
Date Range	@September 25, 2025 → September 25, 2025
Days Left	0
Category	.NET MAUI
Progress Bar	0
Task Left	0/1 Tasks are completed.
Done	☐
Created time	@September 18, 2025 1:31 PM
Last edited time	@October 11, 2025 7:35 PM
Tasks	Task 01

Definición de la idea, alcance y requisitos

- Nombre del proyecto
- Objetivo
- Alcance
- Requisitos funcionales preliminares
- Modelo de dominio
- Diagrama de clases / modelo de clases
- DER (Diagrama de Entidad-Relación)
- Creación de la base de datos física
- Modelo de acceso a datos
- Lógica de negocio (Business Logic) - View Model
- Interfaz de usuario (UI / Presentación)
- UML completo del sistema

Definición de la idea, alcance y requisitos

Nombre del proyecto

Gestión de Catálogo de Artículos (GCA)

Objetivo

Desarrollar una aplicación de escritorio que permita administrar artículos de un catálogo de un comercio genérico. La aplicación está dirigida a usuarios internos del comercio (ej. empleados administrativos) que deben mantener actualizado el catálogo de productos para que luego pueda ser consumido por otros sistemas (web, mobile, e-commerce).

Alcance

- CRUD de artículos (alta, baja, modificación, listado, detalle).
- CRUD de marcas y categorías.
- Búsqueda de artículos por distintos criterios.
- Gestión de imágenes múltiples por artículo.
- Persistencia en SQL Server.
- Validaciones de datos mediante DataAnnotations.
- Separación en capas (UI, lógica de negocio, acceso a datos).
- Implementación del patrón MVVM en la capa de presentación.

No incluye (fuera de alcance):

- Integración real con e-commerce, apps o webs.
- Procesos de facturación, stock o ventas.
- Seguridad avanzada (roles, autenticación).
- UI estética avanzada (centrado en la arquitectura y el código limpio).

Requisitos funcionales preliminares

1. El usuario puede **listar** artículos con sus datos principales.
2. El usuario puede **buscar** artículos por nombre, categoría o marca.

3. El usuario puede **agregar** un artículo con todos sus datos (código, nombre, descripción, marca, categoría, precio, imágenes).
4. El usuario puede **modificar** los datos de un artículo existente.
5. El usuario puede **eliminar** un artículo.
6. El usuario puede **ver** el **detalle** de un artículo, incluyendo imágenes.
7. El usuario puede **administrar** las **marcas** disponibles.
8. El usuario puede **administrar** las **categorías** disponibles.

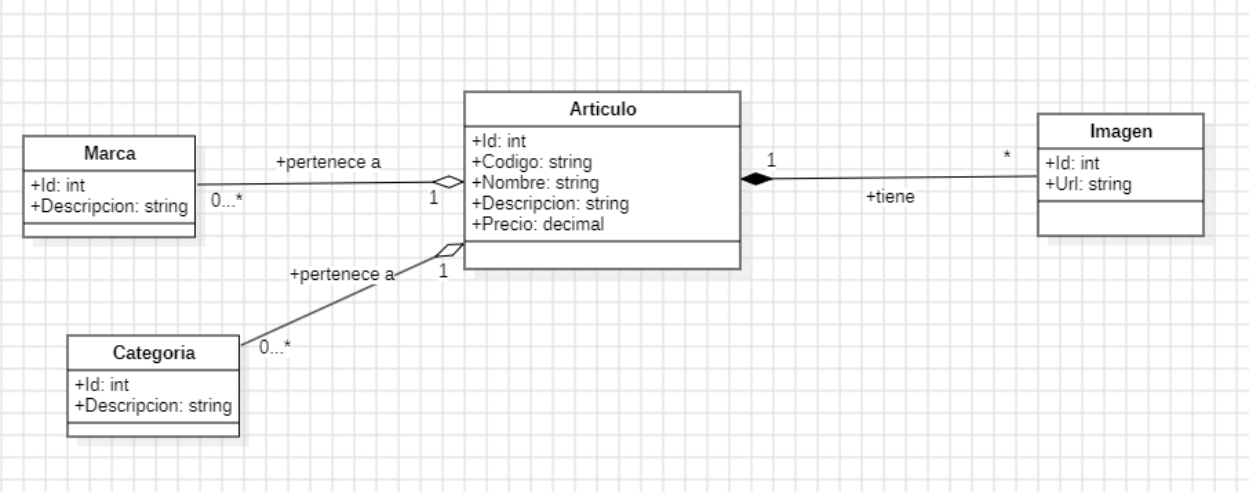
Modelo de dominio

1. **Artículo**
    a. Id
    b. Código
    c. Nombre
    d. Descripción
    e. Precio
    f. Puede tener varias Imágenes
    g. Pertenece a una Marca
    h. Pertenece a una Categoría
2. **Imagen**
    a. Id
    b. Url
    c. Pertenece a un Artículo
3. **Marca**
    a. Id
    b. Descripción
4. **Categoría**

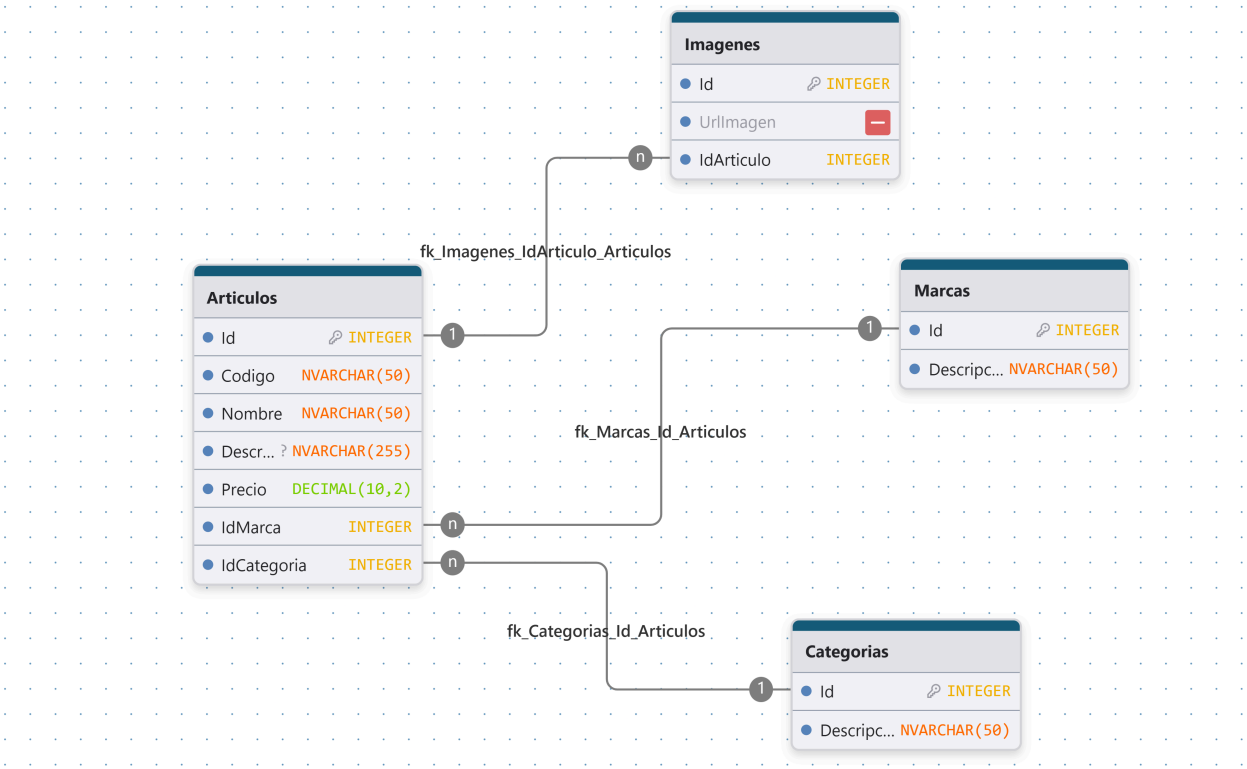
a. Id

b. Descripción

Diagrama de clases / modelo de clases



DER (Diagrama de Entidad-Relación)



Creación de la base de datos física

```
CREATE DATABASE BDCatalogo;

GO

USE BDCatalogo;

GO

CREATE TABLE Marcas (
    IdMarca INT PRIMARY KEY IDENTITY(1, 1),
    Descripcion NVARCHAR(50) NOT NULL
);


GO
```

```
CREATE TABLE Categorias (
    IdCategoria INT PRIMARY KEY IDENTITY(1, 1),
    Descripcion NVARCHAR(50) NOT NULL
);

GO

CREATE TABLE Articulos (
    IdArticulo INT PRIMARY KEY IDENTITY(1, 1),
    Codigo NVARCHAR(50) NOT NULL,
    Nombre NVARCHAR(50) NOT NULL,
    Descripcion NVARCHAR(255),
    Precio Decimal(10, 2) NOT NULL DEFAULT 0,
    IdMarca INT NOT NULL,
    IdCategoria INT NOT NULL,
    CONSTRAINT FK_Articulos_Marcas FOREIGN KEY(IdMarca) REFERENCES
Marcas(IdMarca),
    CONSTRAINT FK_Articulos_Categorias FOREIGN KEY(IdCategoria) REFERE
NCES Categorias(IdCategoria)
);


GO

CREATE TABLE Imagenes (
    IdImagen INT PRIMARY KEY IDENTITY(1, 1),
    UrlImagen NVARCHAR(255) NOT NULL,
    IdArticulo INT NOT NULL,
    CONSTRAINT FK_Imagenes_Articulos FOREIGN KEY(IdArticulo) REFERENC
ES Articulos(IdArticulo)
);
```

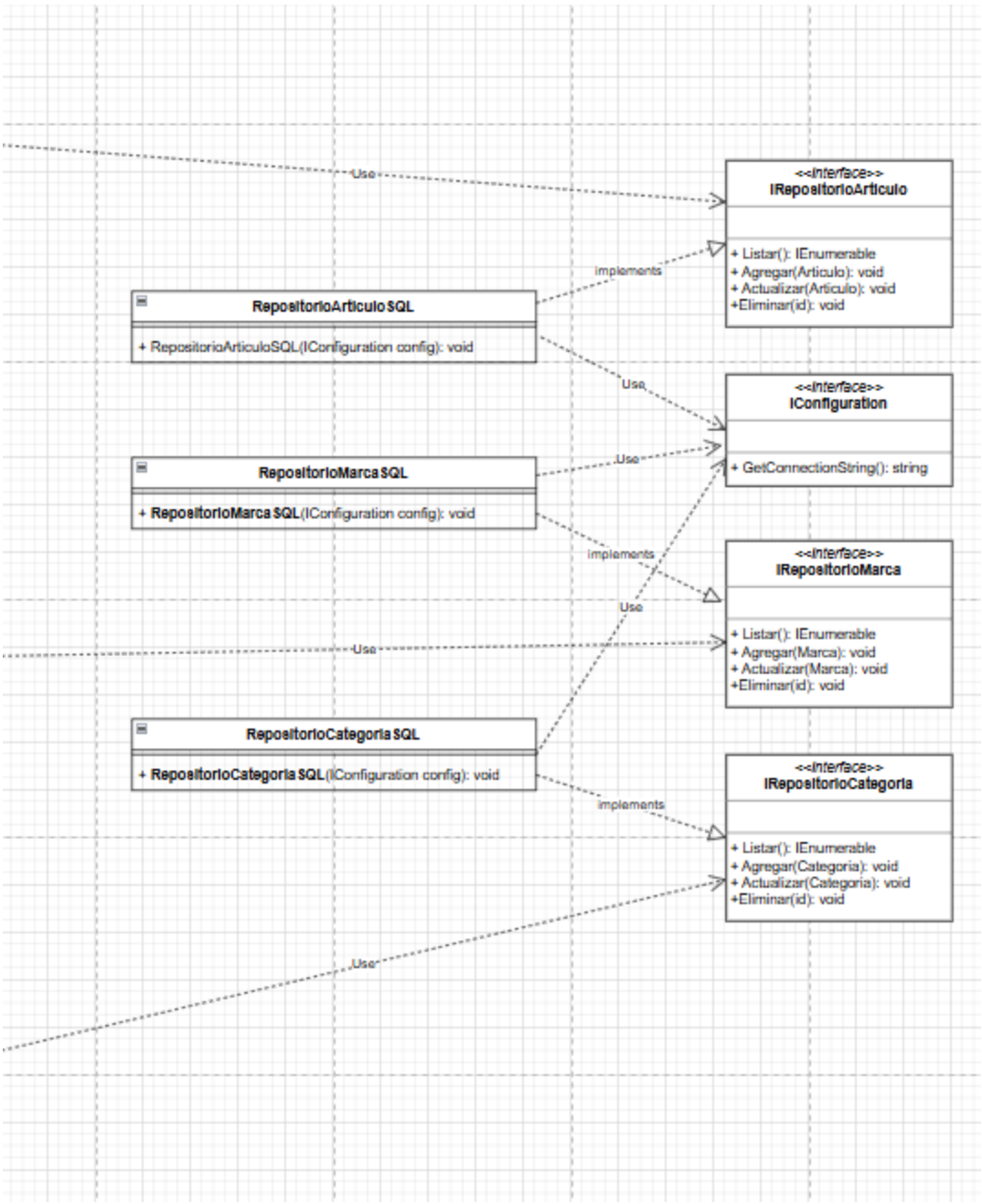
Modelo de acceso a datos

En esta fase se detalla el modelo de cómo se espera que será el acceso a los datos dentro del sistema. Todo parte de una clase abstracta base:

RepositorioBase: Esta clase contiene un campo protegido con la cadena de conexión a la base de datos. Es abstracta para que solo puedan heredar las clases hijas.

Las clases hijas serán las que se conectan al repositorio en sí, es decir, las que manejan la conexión a la base de datos. Estas clases implementan una interfaz.

Cada interfaz indica el contrato que deben cumplir cada clase, esto es así debido a que se quiere lograr el máximo desacoplamiento para facilitar las pruebas, así como la escalabilidad a futuro en caso de querer cambiar alguna implementación esto se puede hacer de manera rápida sin tener que cambiar muchos lugares.



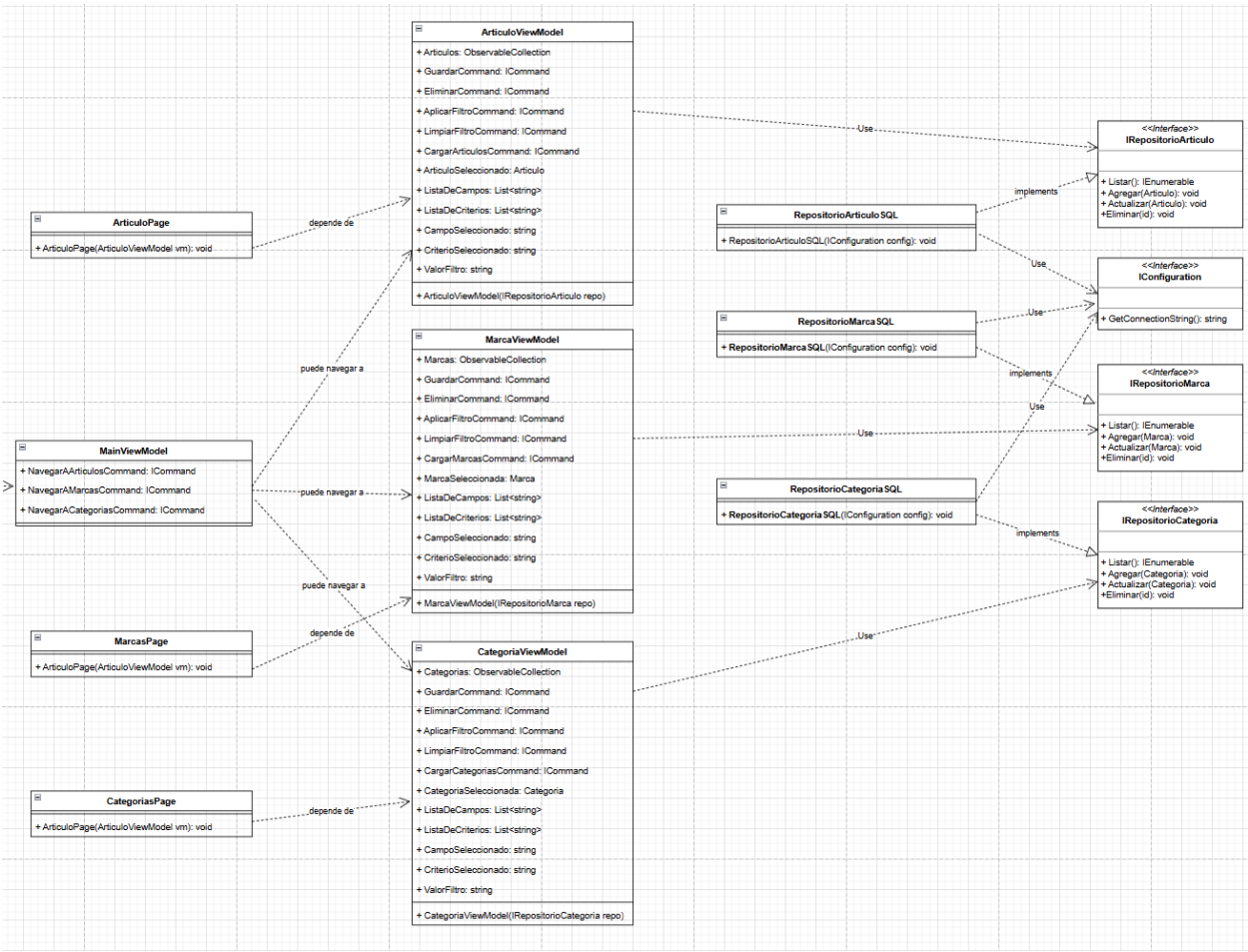
Lógica de negocio (Business Logic) - View Model

Esta capa conecta el Modelo con la Vista. **No** conoce la interfaz de usuario (UI) directamente. Sus responsabilidades son:

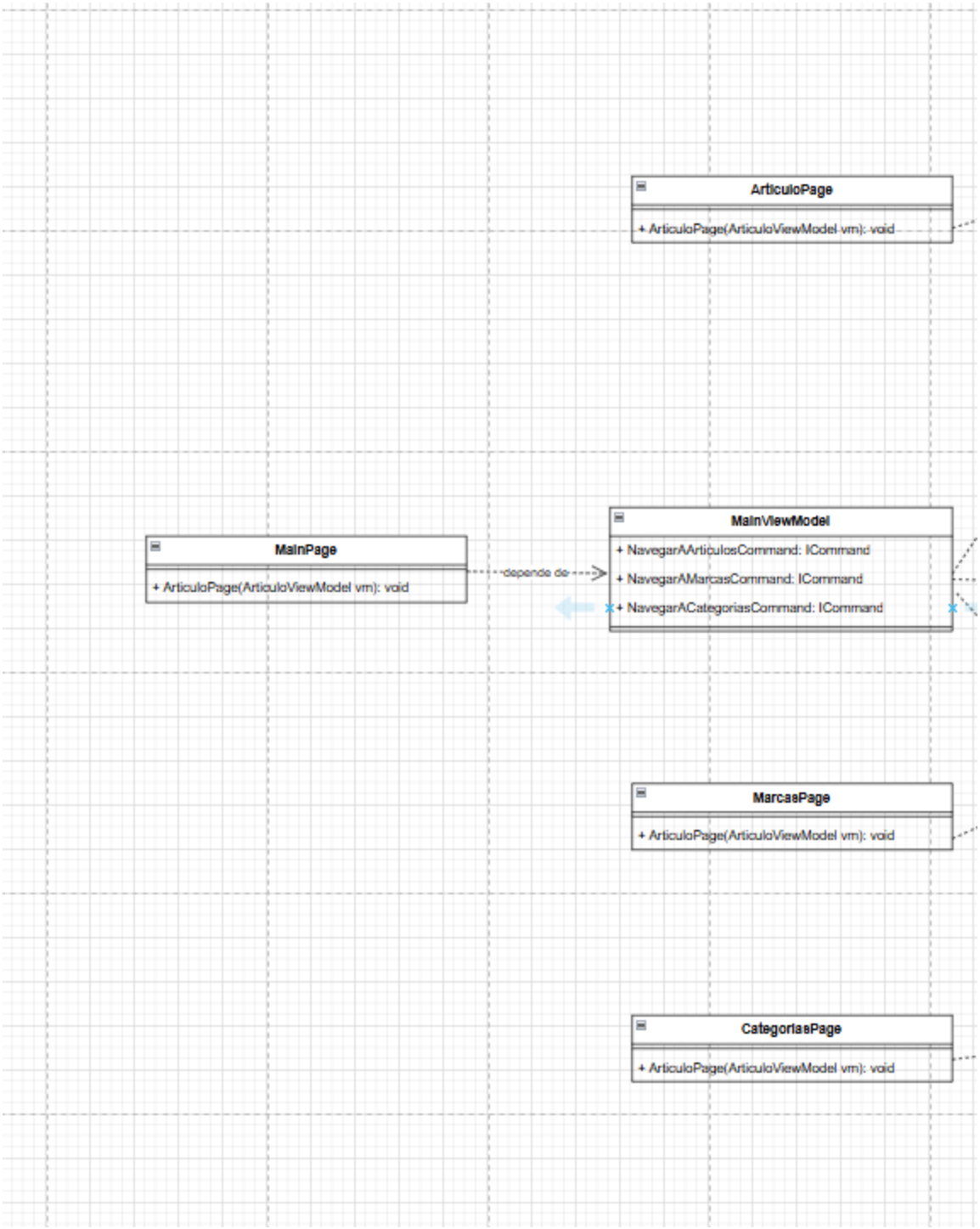
- Mantener el estado de la vista (ej. una lista de artículos para mostrar).
- Exponer datos del Modelo a la Vista a través de propiedades públicas.

- Exponer acciones que el usuario puede realizar a través de **Comandos** (ej. un comando para guardar un artículo).
- Interactuar con la capa de datos (Repositorios) para obtener o guardar datos.

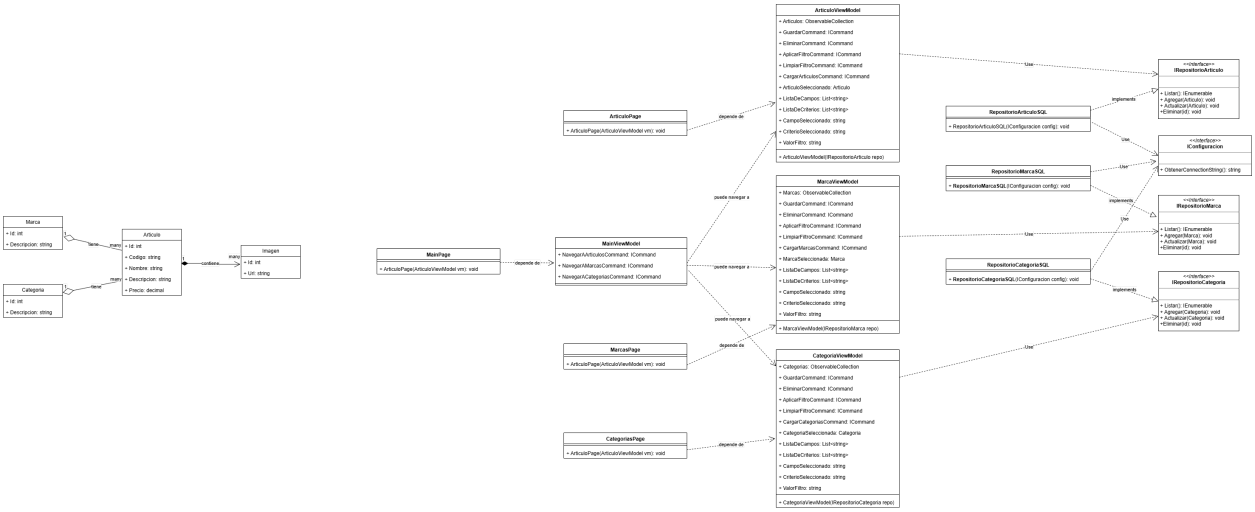
View Model	Vista asociada	Repositorio	Función principal
ArticuloViewModel	ArticuloPage	IArticuloRepository	Expone las propiedades y Comandos que necesita la página para manipular los artículos
MarcaViewModel	MarcaPage	IMarcaRepository	Igual que Artículo, pero para Marcas
CategoriaViewModel	CategoriaPage	ICategoriaRepository	Igual que Artículo, pero para Categorías



Interfaz de usuario (UI / Presentación)



UML completo del sistema




```
classDiagram
    direction LR

    %% ========================================
    %% 1. MODELO DE DOMINIO (Sin Cambios)
    %% ========================================
    class Articulo {
        + Id: int
        + Codigo: string
        + Nombre: string
        + Descripcion: string
        + Precio: decimal
    }
    class Marca {
        + Id: int
        + Descripcion: string
    }
    class Categoria {
        + Id: int
        + Descripcion: string
    }
    class Imagen {
        + ImagenId: int
```

```
    + UrlImagen: string
}
%% --- Relaciones de Dominio ---
Marca "1" o-- "many" Articulo : tiene (Agregación)
Categoria "1" o-- "many" Articulo : tiene (Agregación)
Articulo "1" *-- "many" Imagen : contiene (Composición)

%% ==========================================
%% 2. ABSTRACCIONES (Interfaces)
%% ==========================================
class IConfiguration {
    <<interface>>
    + GetConnectionString(name): string
}
class IRepositorioArticulo {
    <<interface>>
    + Listar(): IEnumerable<Articulo>
    + Agregar(Articulo): int
    + Actualizar(Articulo): void
    + Eliminar(id): void
}
class IRepositorioMarca {
    <<interface>>
    + Listar(): IEnumerable<Marca>
    + Agregar(Marca): int
    + Actualizar(Marca): void
    + Eliminar(id): void
}
class IRepositorioCategoria {
    <<interface>>
    + Listar(): IEnumerable<Categoria>
    + Agregar(Categoria): int
    + Actualizar(Categoria): void
    + Eliminar(id): void
}
```

```
%% ========================================
%% 3. LÓGICA DE LA VISTA (ViewModels)
%% ========================================
class MainViewModel {
    + NavigateToArticulosCommand: ICommand
    + NavigateToMarcasCommand: ICommand
    + NavigateToCategoriasCommand: ICommand
}
class ArticuloViewModel {
    + ArticuloViewModel(IRepositorioArticulo repo)
    + Articulos: ObservableCollection<Articulo>
    + ArticuloSeleccionado: Articulo
    + ListaDeCampos: List<string>
    + ListaDeCriterios: List<string>
    + CampoSeleccionado: string
    + CriterioSeleccionado: string
    + ValorFiltro: string
    + CargarArticulosCommand: ICommand
    + GuardarCommand: ICommand
    + EliminarCommand: ICommand
    + AplicarFiltrosCommand: ICommand
    + LimpiarFiltrosCommand: ICommand
}
class MarcaViewModel {
    + MarcaViewModel(IRepositorioMarca repo)
    + Marcas: ObservableCollection<Marca>
    + MarcaSeleccionada: Marca
    + ListaDeCampos: List<string>
    + ListaDeCriterios: List<string>
    + CampoSeleccionado: string
    + CriterioSeleccionado: string
    + ValorFiltro: string
    + CargarMarcasCommand: ICommand
    + GuardarCommand: ICommand
    + EliminarCommand: ICommand
    + AplicarFiltrosCommand: ICommand
```

```
    + LimpiarFiltrosCommand: ICommand
}
class CategoriaViewModel {
    + CategoriaViewModel(IRepositorioCategoria repo)
    + Categorias: ObservableCollection<Categoria>
    + CategoriaSeleccionada: Categoria
    + ListaDeCampos: List<string>
    + ListaDeCriterios: List<string>
    + CampoSeleccionado: string
    + CriterioSeleccionado: string
    + ValorFiltro: string
    + CargarCategoriasCommand: ICommand
    + GuardarCommand: ICommand
    + EliminarCommand: ICommand
    + AplicarFiltrosCommand: ICommand
    + LimpiarFiltrosCommand: ICommand
}

%% ========================================
%% 4. IMPLEMENTACIONES (Clases Concretas)
%% ========================================
class MainPage {
    + MainPage(MainViewModel vm)
}
class ArticuloPage {
    + ArticuloPage(ArticuloViewModel vm)
}
class MarcaPage {
    + MarcaPage(MarcaViewModel vm)
}
class CategoriaPage {
    + CategoriaPage(CategoriaViewModel vm)
}
class RepositorioArticuloSQL {
    + RepositorioArticuloSQL(IConfiguration config)
}
```

```
class RepositorioMarcaSQL {
   + RepositorioMarcaSQL(IConfiguration config)
}
class RepositorioCategoriaSQL {
   + RepositorioCategoriaSQL(IConfiguration config)
}

%% ========================================
%% RELACIONES ESTRUCTURALES (Adaptadas a MVVM)
%% ========================================

%% --- Implementación de Interfaces ---
RepositorioArticuloSQL --|> IRepositorioArticulo : implements
RepositorioMarcaSQL --|> IRepositorioMarca : implements
RepositorioCategoriaSQL --|> IRepositorioCategoria : implements

%% --- Dependencia/Uso ---
ArticuloViewModel ..> IRepositorioArticulo : usa
MarcaViewModel ..> IRepositorioMarca : usa
CategoriaViewModel ..> IRepositorioCategoria : usa

MainPage ..> MainViewModel : depende de
ArticuloPage ..> ArticuloViewModel : depende de
MarcaPage ..> MarcaViewModel : depende de
CategoriaPage ..> CategoriaViewModel : depende de

RepositorioArticuloSQL ..> IConfiguration : usa
RepositorioMarcaSQL ..> IConfiguration : usa
RepositorioCategoriaSQL ..> IConfiguration : usa

%% --- Composición/Navegación Lógica ---
MainViewModel ..> ArticuloViewModel : puede navegar a
MainViewModel ..> MarcaViewModel : puede navegar a
MainViewModel ..> CategoriaViewModel : puede navegar a
```