

Propuesta Arquitectura

Product Master

(para Tekton Labs)

Enero 2024

**Diseño y Arquitectura**

Enero 2020

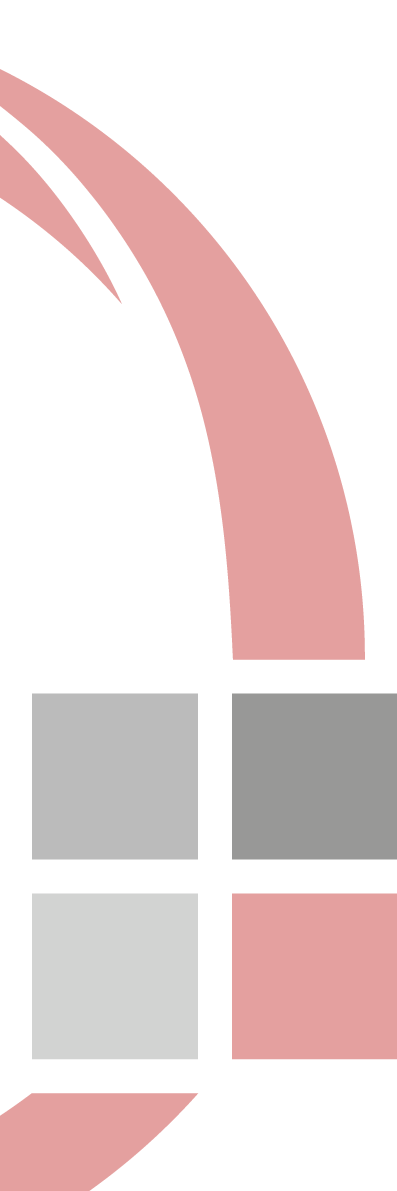
Capa de Dominio

CAPAS

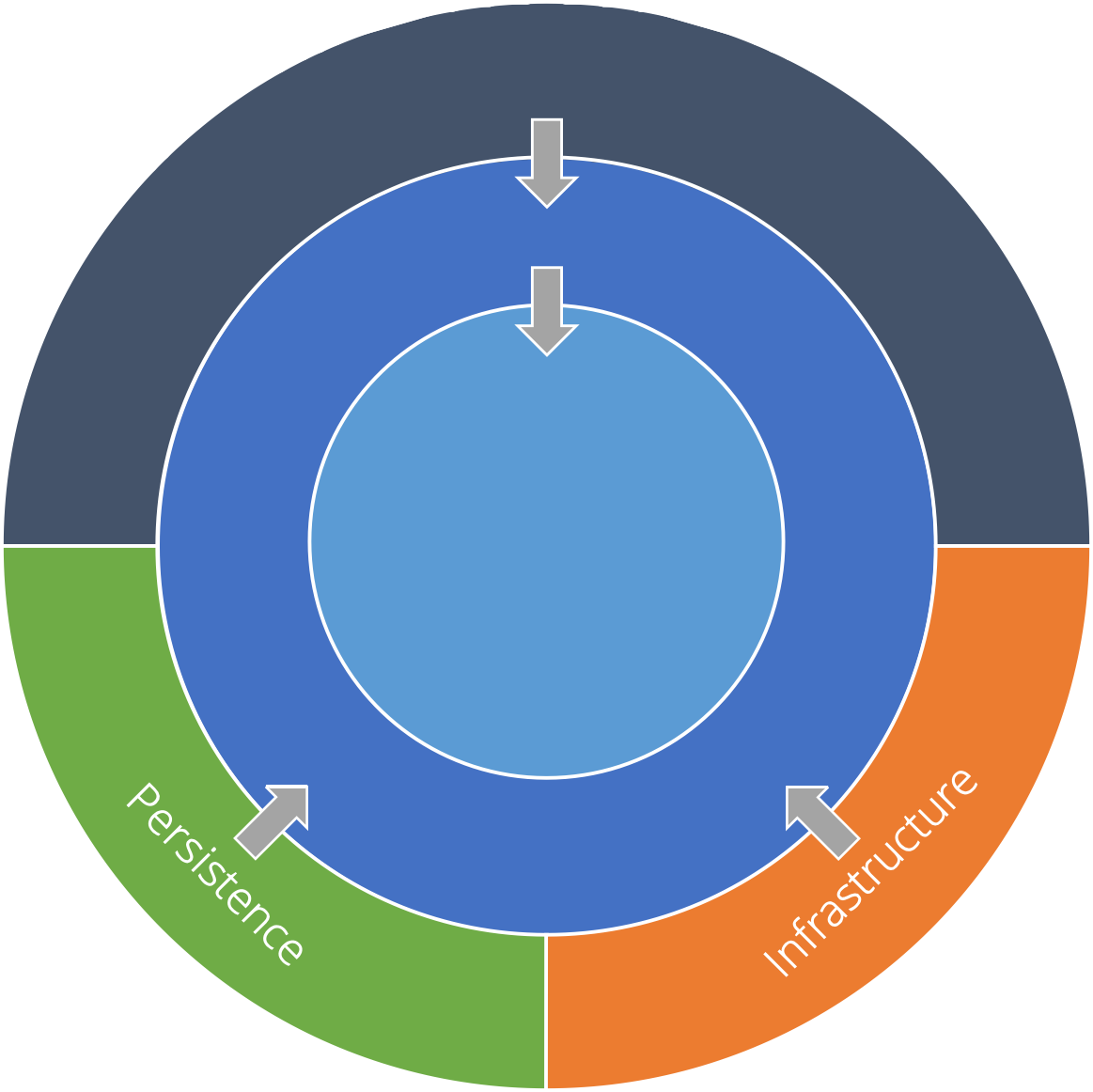
Capa de Aplicación

Capa de Persistencia

Capa de Infraestructura



Vision general



Presentation

Application

Domain

Independiente de otros frameworks

Verificable a través de Pruebas

Independiente de Interfaz Grafica

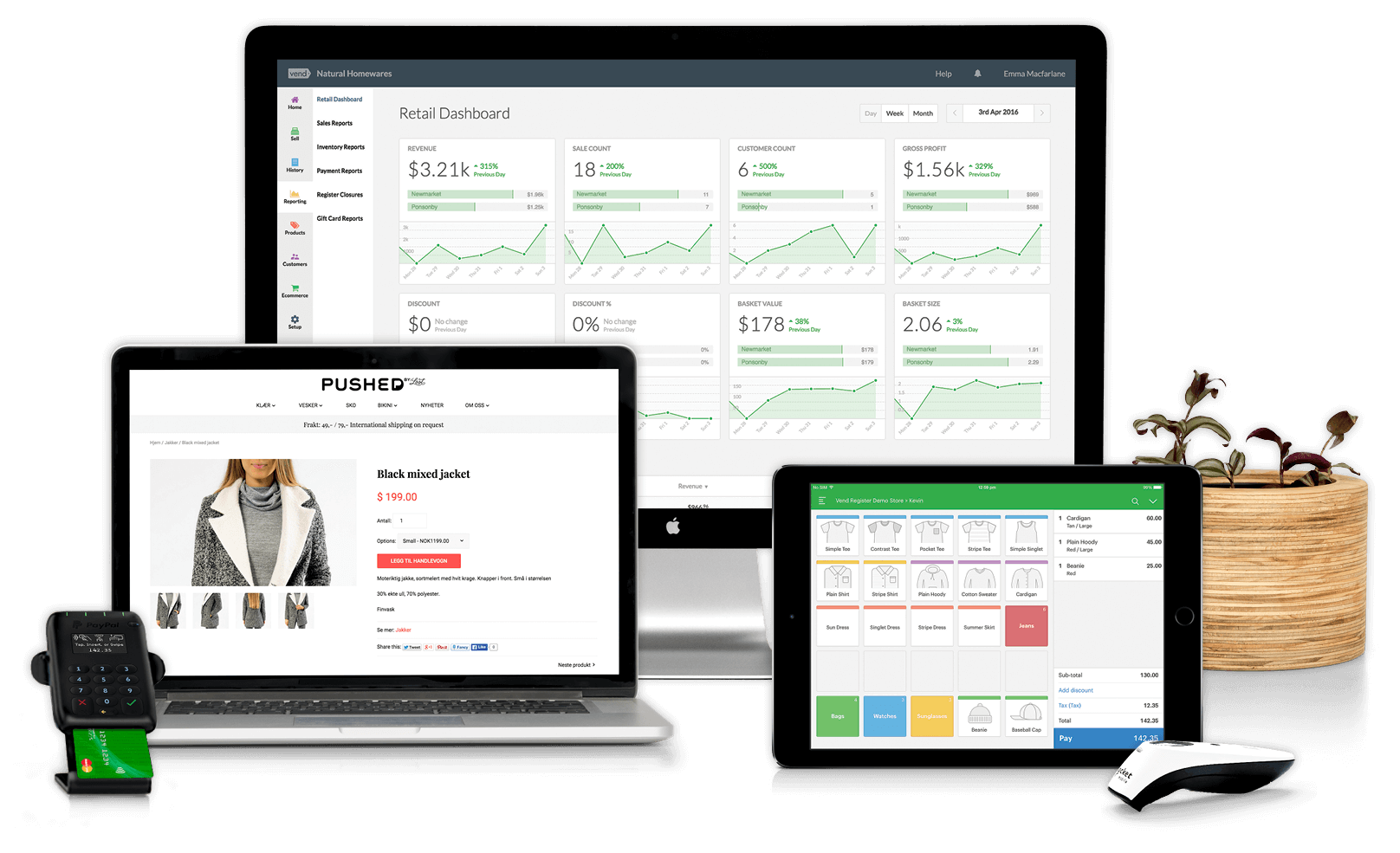
Independiente de la base de datos

Independiente de lo exterior



# Product Master

Varias plataformas

.NET Core

Entity Framework Core Code First

Data Seeding



# Aspectos principales

El dominio contiene la lógica empresarial y los tipos

Puede ser compartida con otras aplicaciones

La capa de aplicación contiene la lógica del negocio y sus tipos

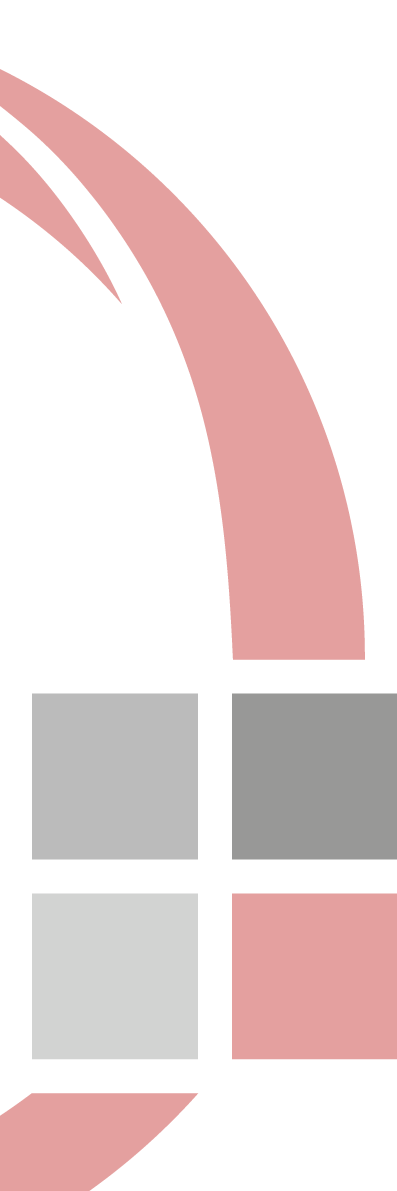
La capa de Infraestructura (y la de Persistencia) contiene todos los aspectos externos

La capa de Presentación y de Infraestructura dependen de Aplicación

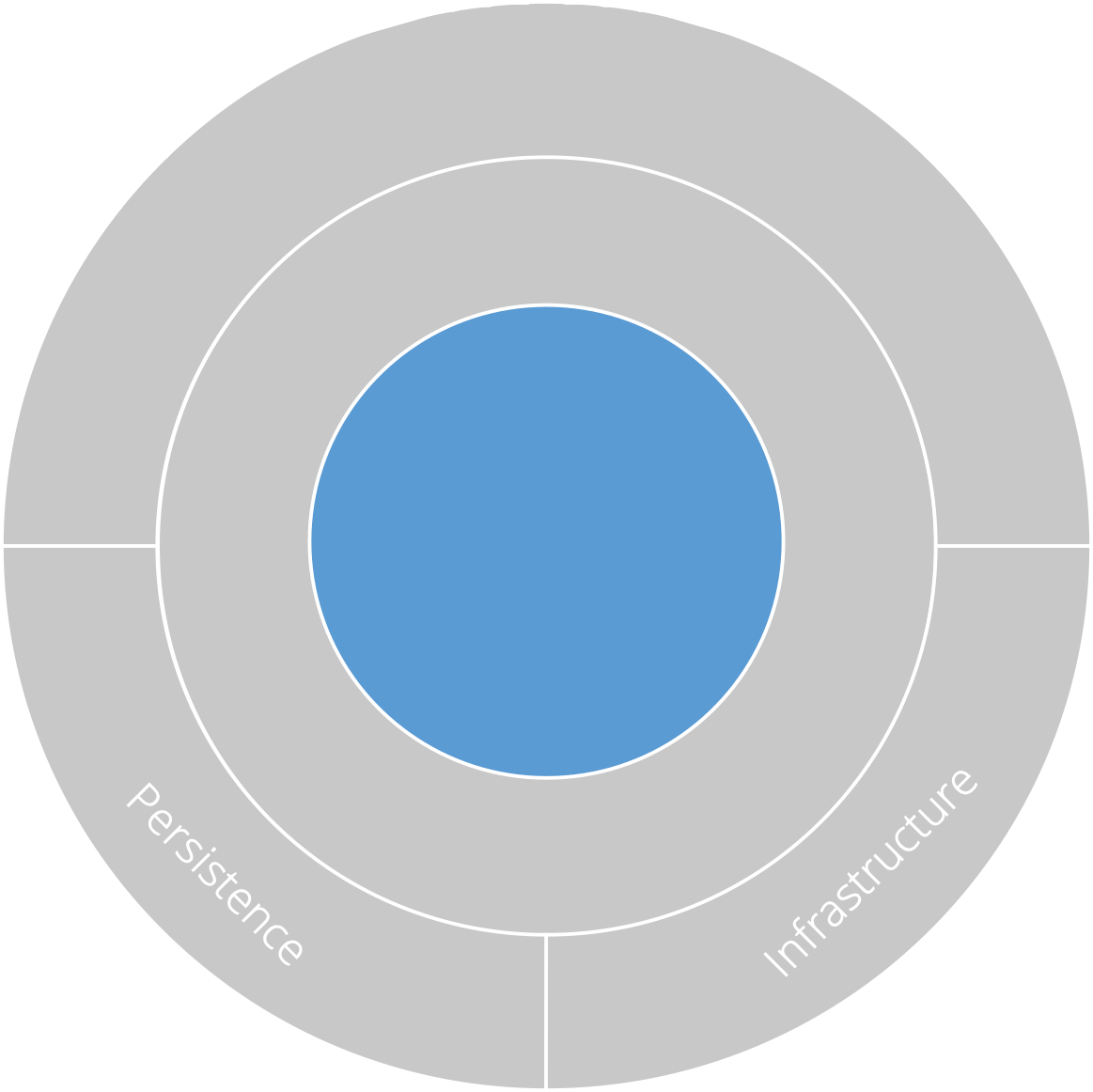
Los componentes de Infraestructura y Presentación pueden ser reemplazados con mínimo esfuerzo

Capa de Domino

CAPAS



# Generalidades



Presentation

Application

Domain

Entidades

Value Objects Enumeraciones Lógica empresarial Excepciones

Aspectos Principales



Evitar el uso de Data annotations.

Usa value Objects cuando es apropiado.

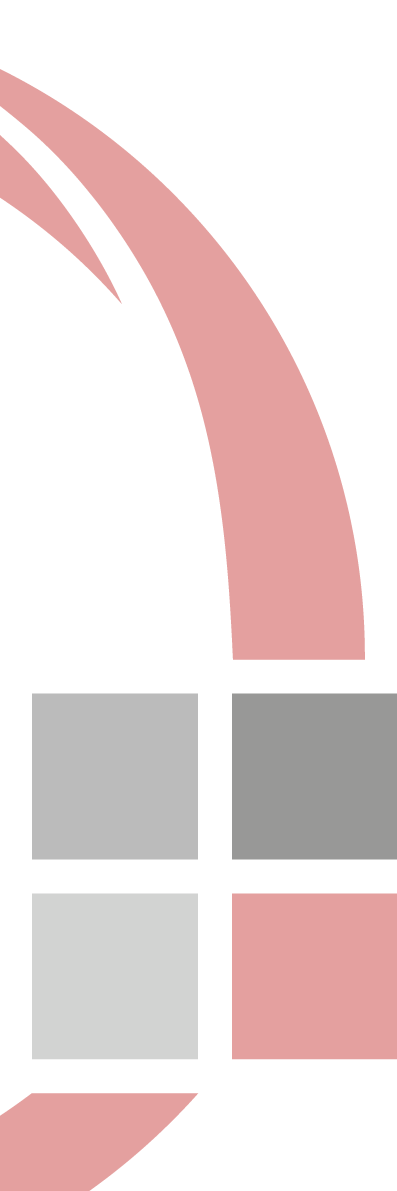
Inicializa todas las colecciones y usa setters privados.

Crear excepciones de dominio personalizadas

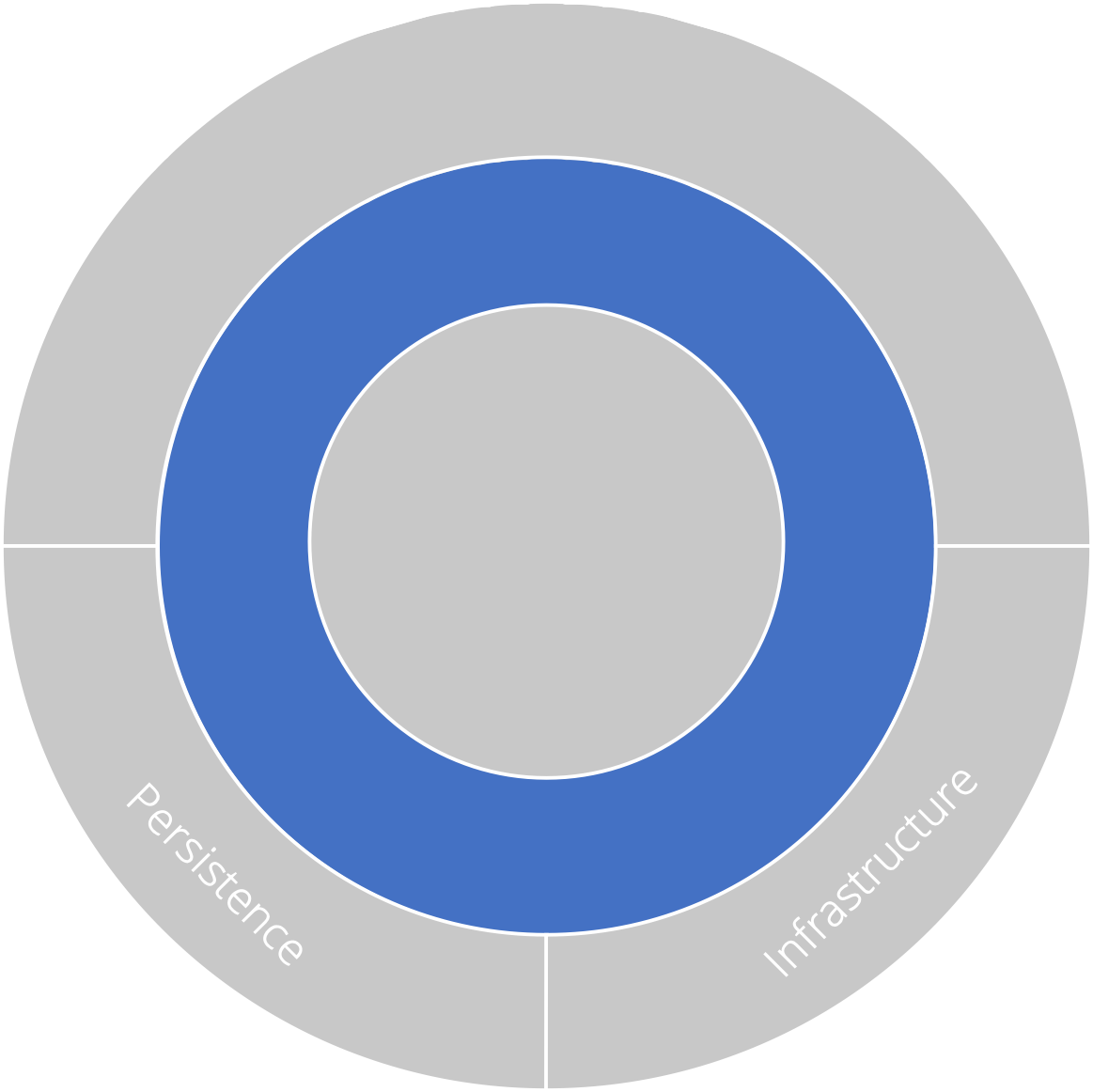
Junto con la capa de Aplicación y Dominio es el Core del aplicativo

CAPAS

Capa de Aplicacion



# Generalidades



Presentation

Application

Domain

Interfaces

Modelos

Lógica de Negocio

Comandos y Queries

Validadores de excepciones



# CQRS Command Query Responsibility Segregation

* Separa lecturas (queries) de escrituras (commands)

* Maximiza el desempeño, escalabilidad, y la simpleza
* Fácil de añadir nuevas características, tan solo añadiendo un nuevo comando o query
* Fácil de mantener, los cambios solo afectan un comando o un query



# MediatR & CQRS

* Define comandos y consultas como peticiones
* La capa de Aplicación es solo una serie de solicitudes y objetos de respuesta
* Capacidad para adjuntar comportamiento adicional antes y / o después de cada solicitud, por ejemplo. registro, validación, almacenamiento en caché, autorización, etc.



# Aspectos Principales

Usar CQRS + MediatR simplifica el diseño general

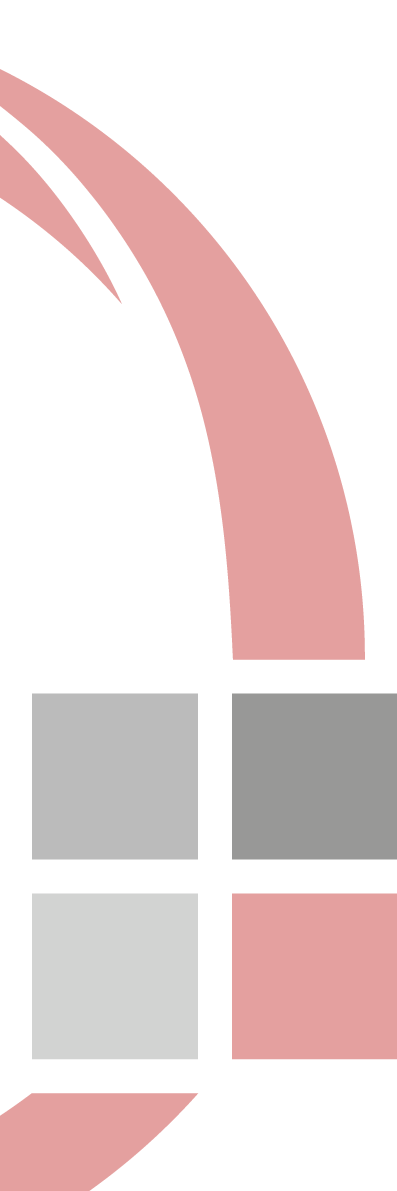
Fluent Validation is útil para la validación de escenarios

MediatR simplifica aspectos transversales

Independencia de aspectos de infraestructura y de acceso a datos

CAPAS

Persistence Layer



# Generalidades

DbContext Migraciones Configuraciones Seeding Abstracciones



# Patrones de Unit of Work y Repository

En .Net Core estos patrones no siempre son la mejor opción,

porque:

EF Core aísla el código de los cambios en la de Datos

El DbContext actúa como unit of work

dB Set actúa como repositorio

EF Core tiene características para unit testing sin repositorios



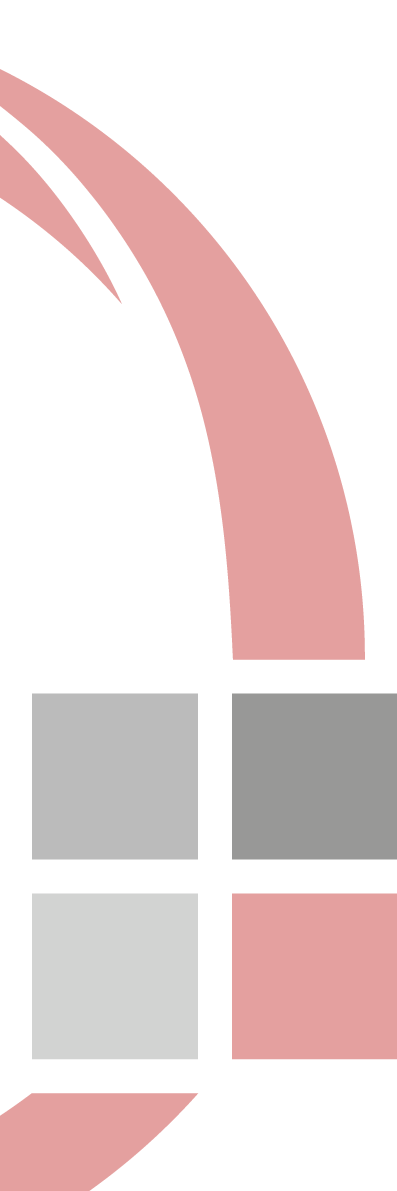
# Aspectos principales

Independencia de la base de datos

Usar la configuración del API Fluent sobre Data Annotations

Preferir convenciones sobre configuración

Aplica automáticamente todas las configuraciones de tipo de entidad.

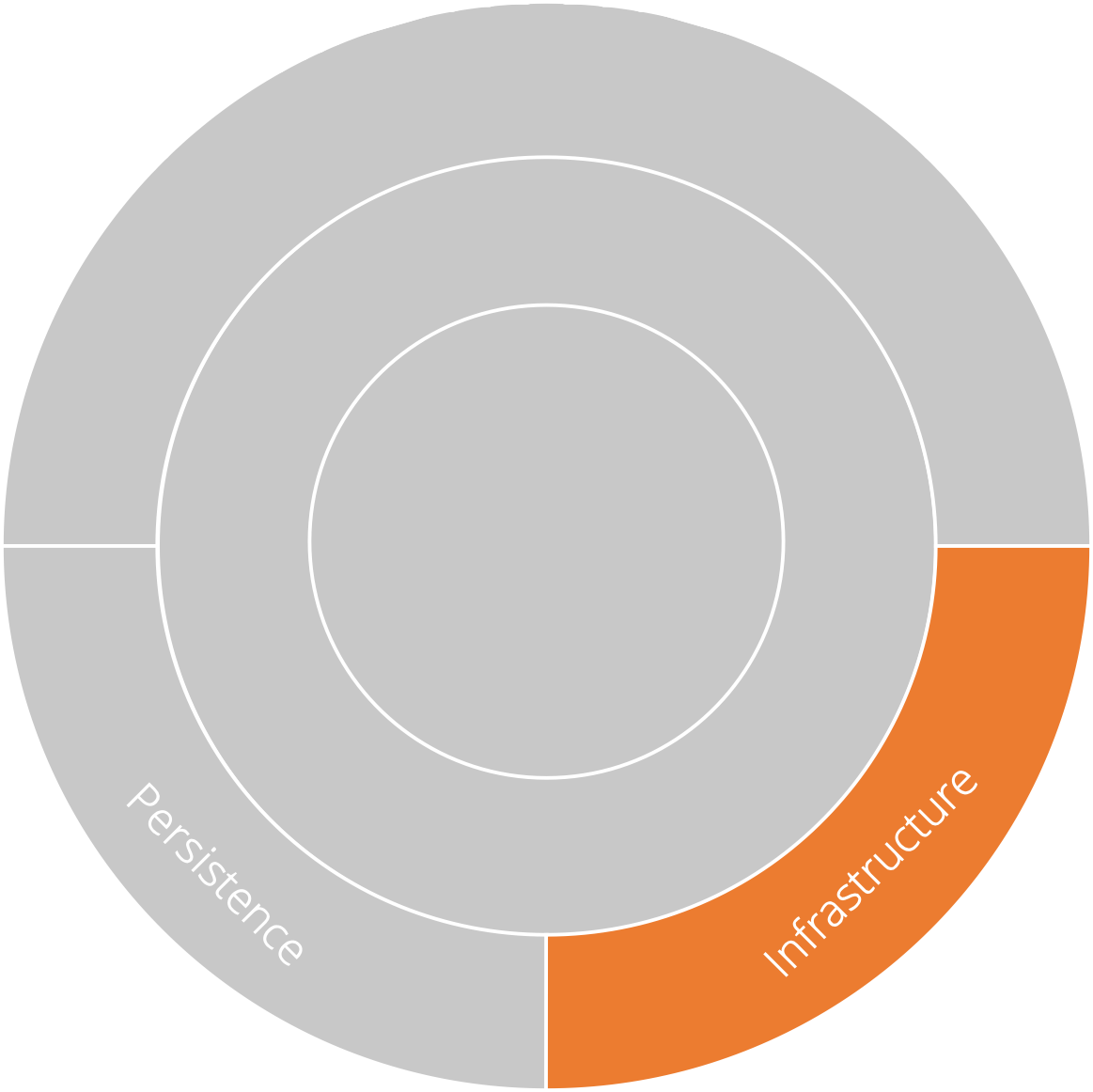


CAPAS

Infrastructure Layer



# Generalidades



Presentation

Application

Domain

Implementaciones, tales como:

* Clientes API: RestSharp
* Sistema de archivos
* Lazy cache
* Fake Data:Bogus
* Cualquiera otra cosa externa

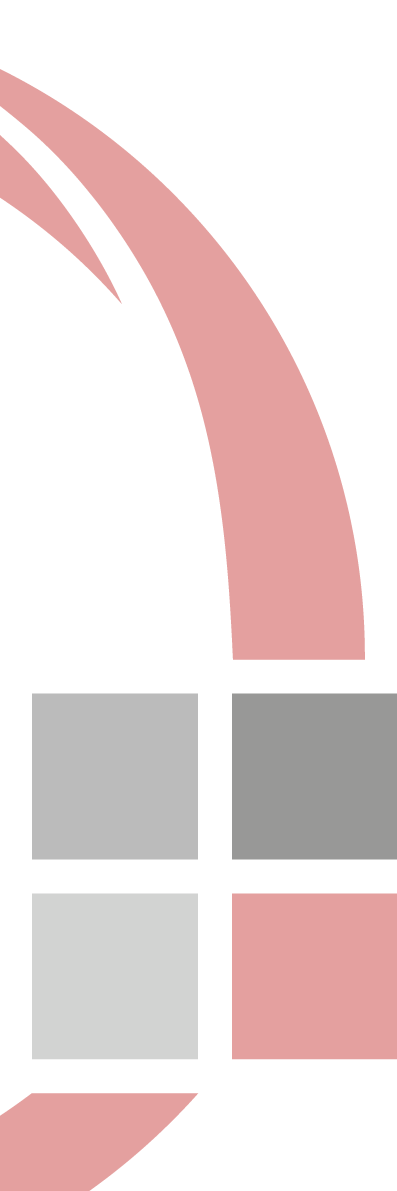


# Aspectos principales

Contiene clases para acceder a recursos externos, como sistemas de archivos, servicios web, SMTP, etc. Implementa abstracciones e interfaces definidas dentro de la capa de aplicación

Ninguna capa depende de la capa de infraestructura, por ejemplo, la de presentación

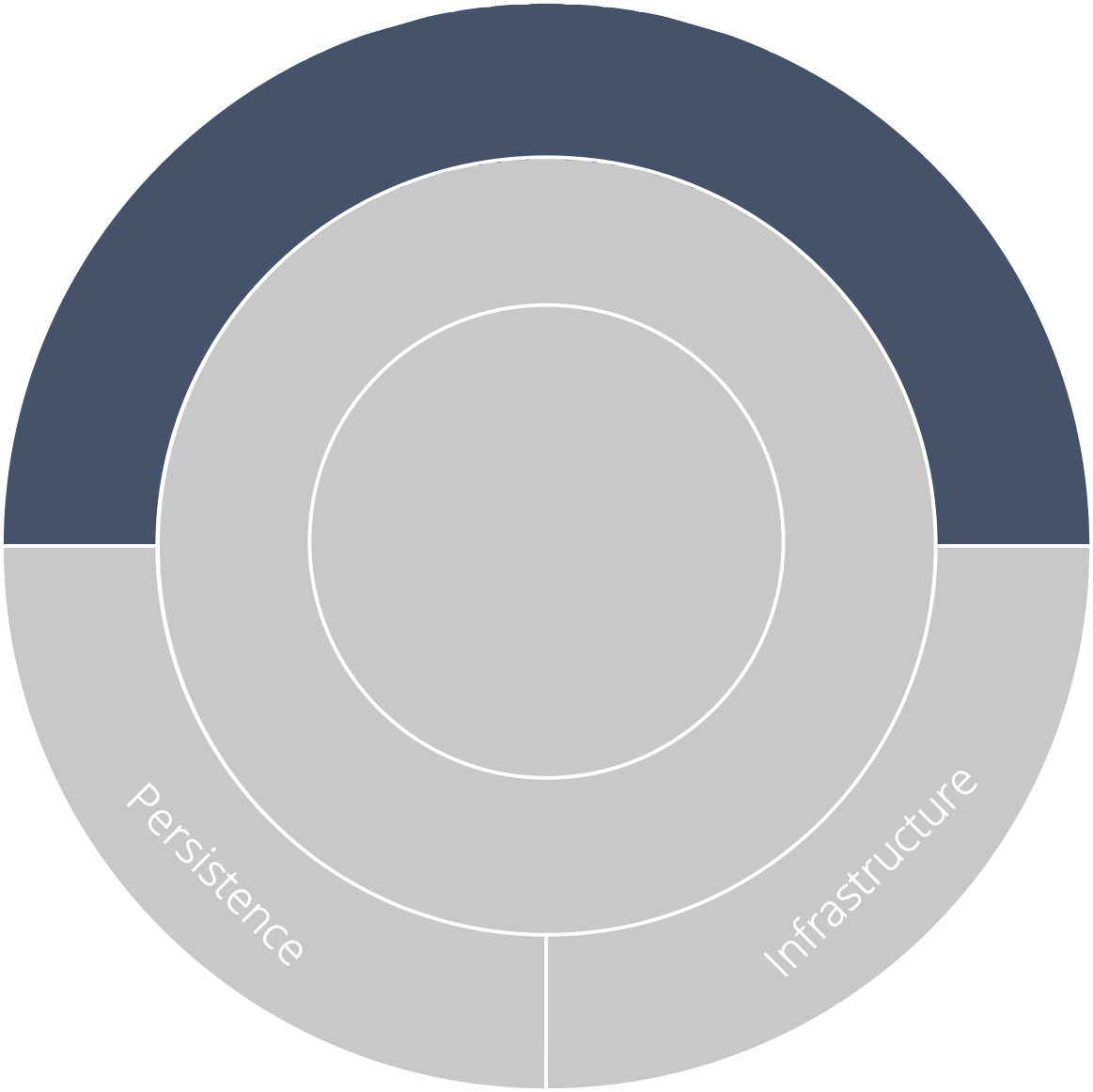
CAPAS



Presentation Layer



# Generalidades



Presentation

Application

Domain

React Native

Web API

Razor Pages MVC

Web Forms



# Aspectos principales

Los endpoints no deben contener ninguna lógica de aplicación.

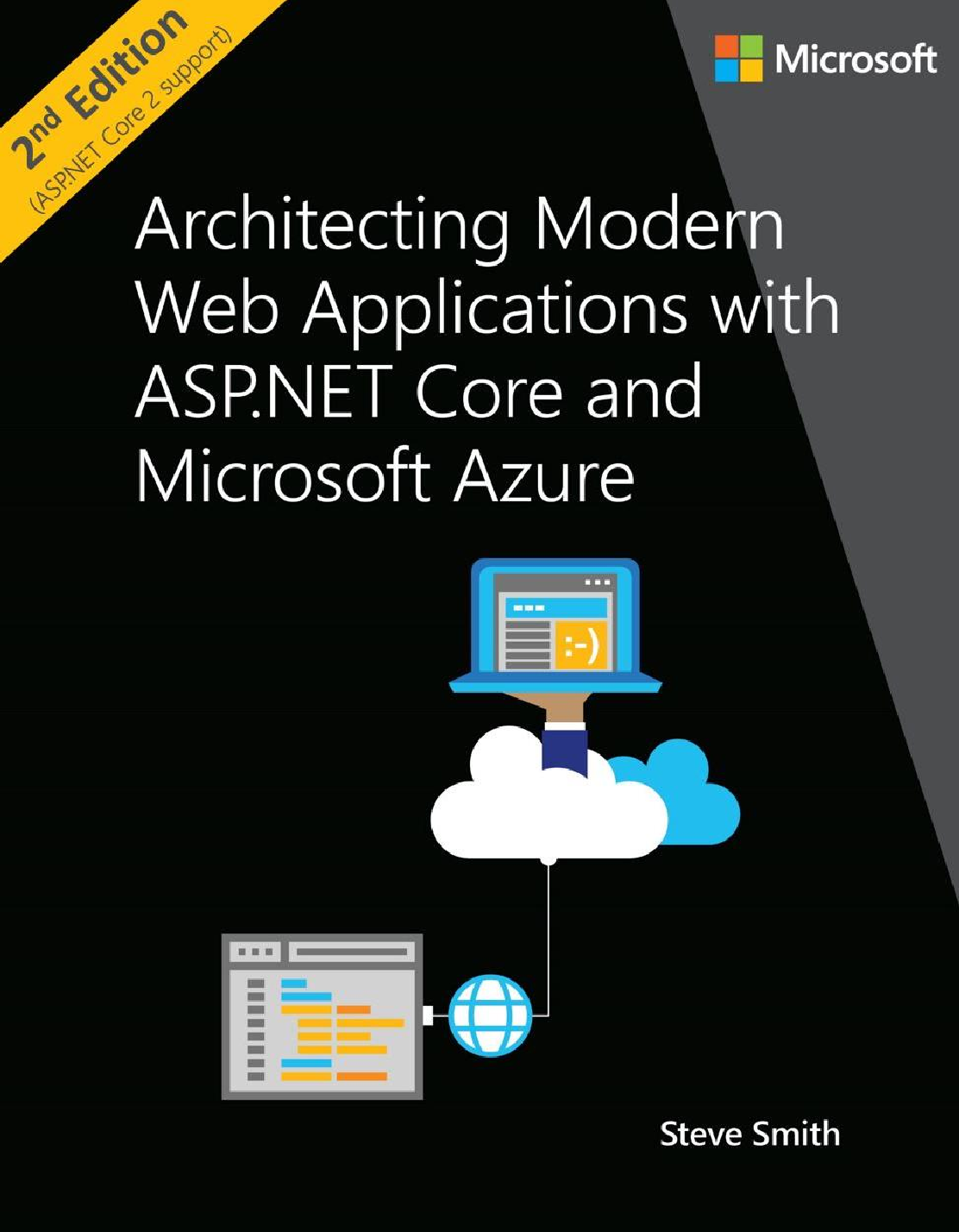
Crear y consumir modelos de vista bien definidos.

Una API abierta cierra la brecha entre el backend y el FrontEnd



Lecturas Recomendadas





Building Monoliths

Clean Architecture

Azure



Building Microservices

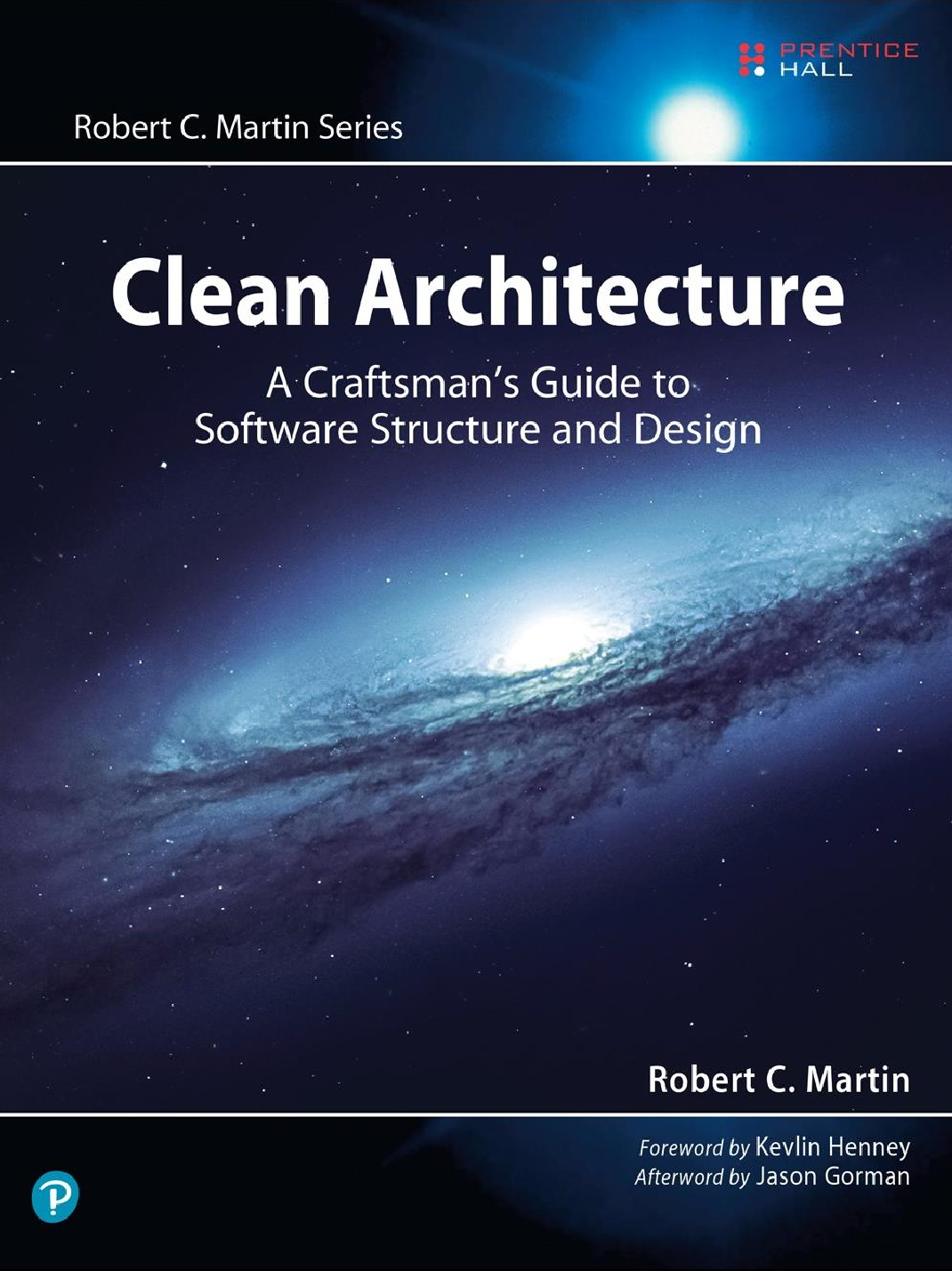
Microservices

Containers

DDD

Azure



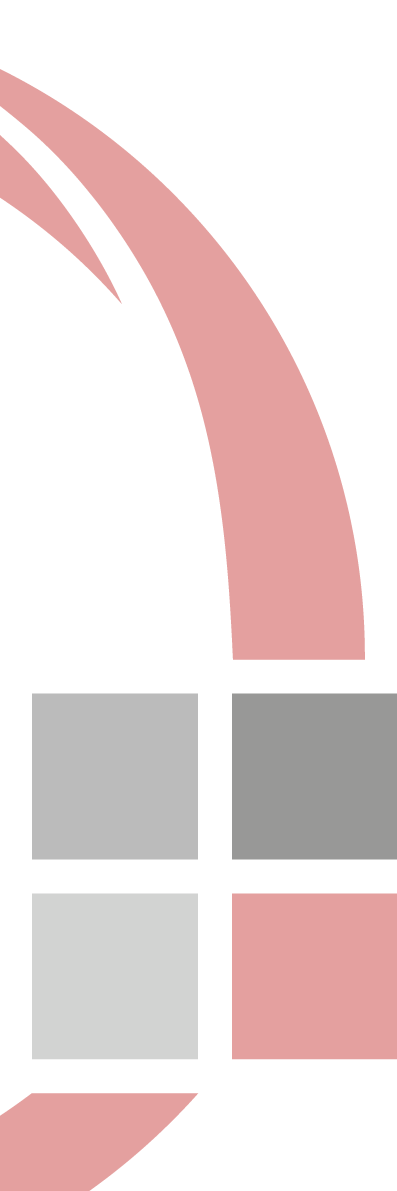




# Clean Architecture

Robert C. Martin







Bogota | Colombia