**Clean Architecture avec CQRS et mediator**

**Liens :**

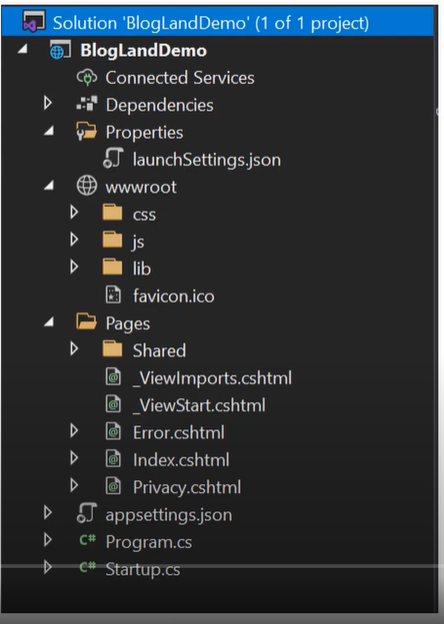
https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architect

**I/ Les types des architectures :**

**1/ All-in-One Architecture : One Single Project : (Architecture tout en un) :**

* Un projet
* Contient tout
* La séparation des préoccupations se fait avec des dossiers
* Bon pour les petits projets.
* Mal de la taille et de la complexité du projet augmentent.

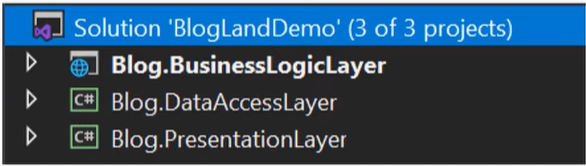
Exemple :

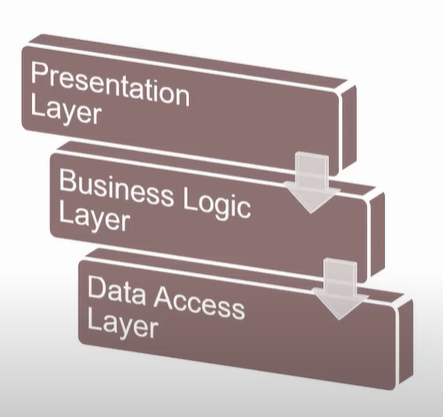


2/ **Layred Architecture :** **Architecture en couche :** **N-Layer Couche :**

* Séparation des préoccupations.
* Réutilisabilité du code DRY : Don’t Repeat YourSelf
* Encapsulation améliorée ‘’Encapsulation : la visibilité ou le niveau de protection des accesseurs(attributs)’’.
* Presistance Agnostic (Indiffèrent à la présistance) : cad n’est pas intéressée par la technologie utilisée ‘’PEUT IMPORTE ‘’.
* Maintenable.

Exemple :

****

****

**Désavantages :**

* **Dépendance de haut en bas (les dépendances de tt les couches lier par eux cad sur chaque couche il faut d’ajouter tt les dépendances d’autre couche qui a lié).**
* **Difficile pour l’application de test.**
* **Déployé comme une seule unité.**

**3/ Clean Architecture : CA :**

Ilya Plusieurs nomination pour Architecture Clean comme :

* Hexagonal Architecture
* Ports-And-Adapters
* Onion Architecture

CA Divise en 3 parties importantes :

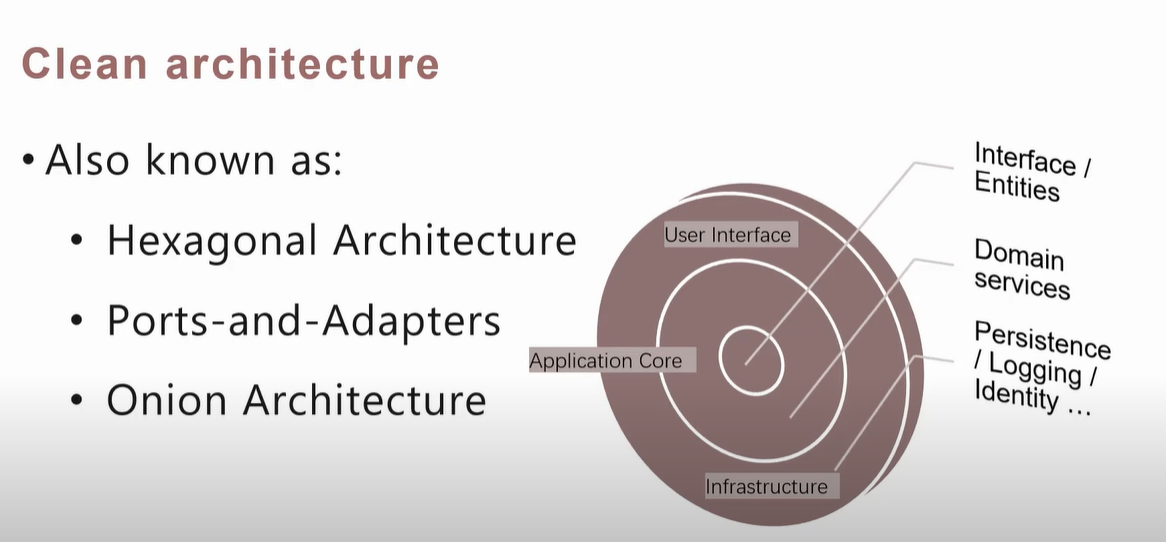
1/ Application Core : Divisé en 2 couches :

- Interface / Entities : (DataBase)

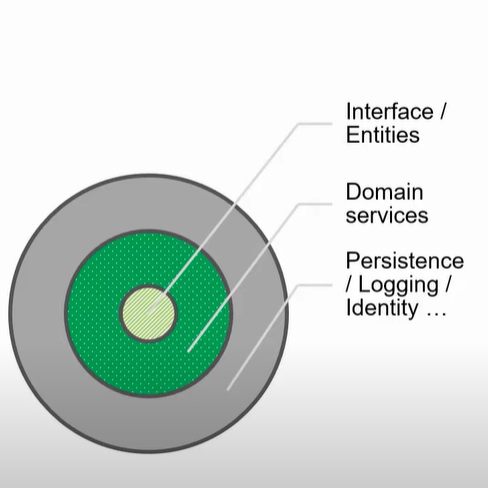
- Domain Service

2/ Infrastructure : Presistence/Logging/ Identity /

3/ User Interface : Présentation

****

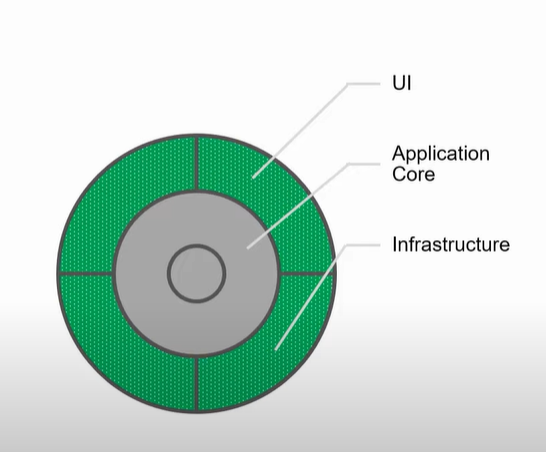
**1/** Application Core : C’est le cœur (noyau) de projet.



* Pas de dépendances externes. (Il faut n’est pas relier avec des dépendances externes mais IL FAUT TT LES DEPENDANCES EXTERNE LIE PAR APPLICATION CORE
* Domain Service
* Abstractions

2/ Infrastructure :

* Data Access / Presistance
* Identity
* File Access
* Logging



3/ User Interface :

* Controllers
* Views
* ViewModels : DTO

**II/ CQRS et MEDIATOR Design Pattern :**

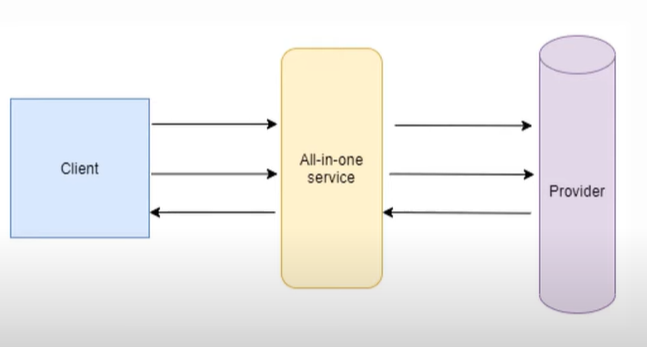
**1/ CQRS Design Pattern :**

**CQRS : C**ommand and **Q**uery **R**esponsibility **S**egregation = Ségrégation des responsabilités de commande et de requête

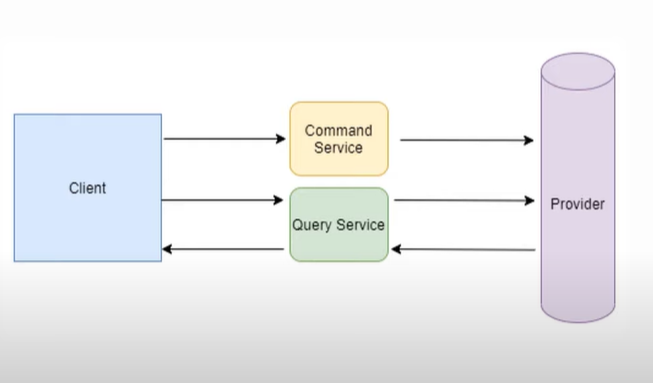
CQRS est un **Design Pattern (Model de conception) architecturale** qui sépare les opérations Read (Query) et update (Command) pour un Data Store (Magasin de données).

**Architecture Classique Vs CQRS Architecture :**

* **Architecture Classique :**
* Ne respecte pas le premier principe de PRINCIPE SOLID : SINGLE RESPONSABILITY
* Tout en un seul service
* Ne respecte pas séparations des préoccupations

****

* **CQRS Architecture :**
* Single Responsability
* Séparations des préoccupations
* Code Modulaire
* Facile Maintenance, Extend, Test
* Scalable

****

**2/ MEDIATOR :**

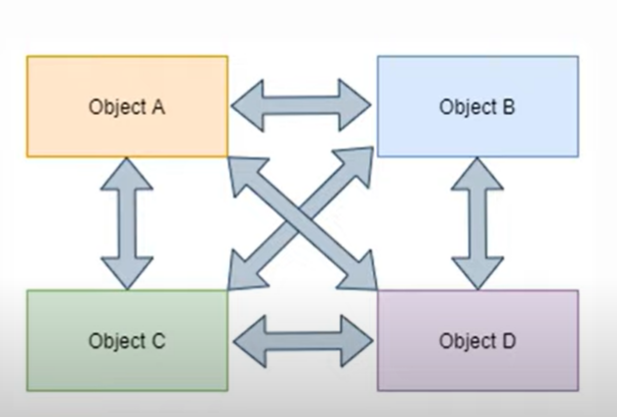
MEDIATOR Design pattern (Model de conception) qui permet à plusieurs objets de communiquer entre eux sans connaître la structure de l'autre.

⬄ Pour gérer la gestion des dépendances

⬄ Coordination les dépendances

**Architecture Classique Vs MEDIATOR Architecture :**

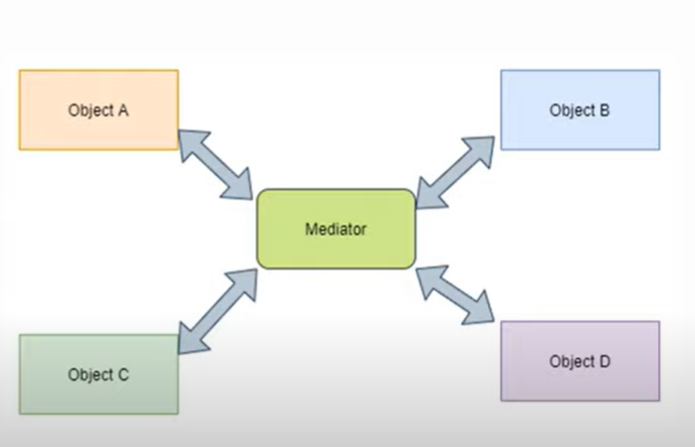
* **Architecture Classique :**
* Fort couplage entre les dépendances.



* **MEDIATOR Architecture :**
* Faible couplage.
* Facile à Maintenir.

NB :

* En .Net en utilise package MediatR.



**III/ Entity Framework Core : EFC :**

EFC c’est un ORM **(Object Relational Mapping)** qui permet aux développeurs .Net de travailler avec une base de données.

**IV/ Swagger :**

Swagger est utilisé pour générer de la documentation pour l'API Web. Il offre également une interface pour explorer et tester les différents endpoints exposés par le service.

**V/ RESTAPI :**

ASP.NET nous permet de créer des services RESTFUL (adopter l'architecture). Nous permettons de créer des services Web accessibles via le protocole HTTP et que nous pouvons consommer par différents clients (Web, Mobile, Desktop).