# **Clean Architecture**

**1. Clean Architecture là gì.**

**Clean Architecture** là một *architectural style* nhằm tách biệt trách nhiệm, bảo vệ domain(entity) khỏi tác động của framework, cơ sở dữ liệu, UI hoặc các phụ thuộc bên ngoài khác. Mục tiêu chính:

* Làm cho **domain** là trung tâm và **không phụ thuộc** vào các thư viện/framework.
* Tăng khả năng **test**, **bảo trì**, **mở rộng** và **thay đổi implementation**.
* Giảm coupling và làm rõ ràng ranh giới trách nhiệm giữa các phần trong hệ thống.

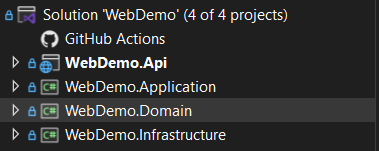
**2. Nguyên tắc cốt lõi**

* **Dependency Rule (Luật phụ thuộc):** phụ thuộc chỉ hướng từ ngoài vào.
* **Dependency Inversion Principle (DIP):** high-level modules **không nên phụ thuộc** low-level modules**;** cả hai nên phụ thuộc **abstraction.**

Diagram of a diagram of a clean architecture

AI-generated content may be incorrect.

**3. Các tầng (layers) & trách nhiệm**



* **Domain**: lõi nghiệp vụ. Chứa Entity, ValueObject, Domain Events. KHÔNG chứa reference (vì đây là lớp core)
* **Application**: Chứa các use-case, DTOs, interfaces (ví dụ IProductRepository) mà Infrastructure sẽ implement, CQRS (command/query handlers).
* **Infrastructure**: implement các interface của Application/Domain (repositories, DbContext), các dịch vụ bên ngoài. Infrastructure được phép reference vào Application & Domain.

# **Liên kết Clean Architecture với CQRS & MediatR**

**1. Lý do kết hợp**

Clean Architecture giúp tách biệt trách nhiệm giữa các tầng, còn CQRS và MediatR giúp tổ chức logic nghiệp vụ rõ ràng và giảm phụ thuộc, Tức là thay vì bình thường sẽ tổ chức theo mô hình là Controller – Service – Repo thì cái thằng CQRS này nó sẽ thay Service thành nó.

**2. CQRS là gì?**

CQRS (Command Query Responsibility Segregation): tách riêng hai loại thao tác dữ liệu:

Command: có nhiệm vụ ghi dữ liệu (tức là mấy câu lệnh insert, update, delete).

Query: có nhiệm vụ đọc dữ liệu (câu lệnh select).

(Command chỉ ghi, Query chỉ đọc).

**3. MediatR là gì?**

**MediatR** là thư viện .NET thực hiện mẫu **Mediator Pattern**, giúp **các lớp không gọi trực tiếp nhau**, mà thông qua một “người trung gian”.

A diagram of a company

AI-generated content may be incorrect.

**Command/Query**: đại diện cho yêu cầu (IRequest).

**Handler**: nơi xử lý logic chính (IRequestHandler).

**Mediator**: tự động tìm đúng handler để thực thi.

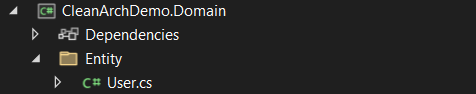
Controller → MediatR → Handler → Repository → Database

Tóm lại, trong Clean Architecture:

* CQRS chia rõ đọc và ghi (Query/Command).
* MediatR giúp controller giao tiếp gián tiếp qua các handler.
* Nhờ đó, controller không chứa logic, application rõ ràng và dễ bảo trì.

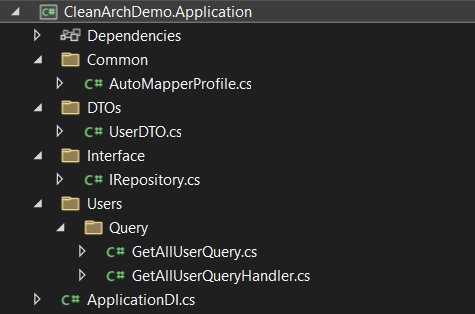
# **Minh họa Clean Architecture + CQRS + MediatR**

T sẽ làm 1 Api lấy hết toàn bộ user (là query trong cqrs).

**BƯỚC 1: DOMAIN LAYER**

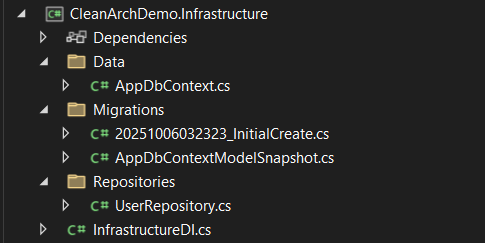
* Xây **nền tảng nghiệp vụ (business core)**.
* Chứa **Entity** và **logic nghiệp vụ thuần túy** (nếu có).
* Không phụ thuộc bất kỳ framework, DB hay lớp ngoài nào.

**BƯỚC 2: APPLICATION LAYER**

Tầng **Application** là cầu nối giữa **Domain** và **Infrastructure**, định nghĩa: 

* DTOs: Chứa các lớp trung gian để truyền dữ liệu giữa tầng Application và bên ngoài (API/UI).
* Interface: Định nghĩa hợp đồng (interface) như IRepository để tầng Infrastructure phải implement.
* Users/Query: Chứa logic xử lý theo mô hình CQRS: Query là hành động “đọc dữ liệu”.
* Common/AutoMapperProfile: Cấu hình AutoMapper để map Entity ↔ DTO tự động.
* ApplicationDI: Chỗ đăng ký Dependency Injection cho toàn tầng Application.

**BƯỚC 3: Infrustructure Layer**



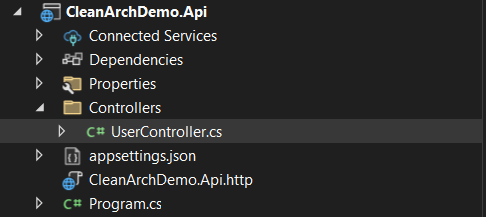
Tầng này **hiện thực (implement)** lại tất cả những gì tầng **Application chỉ “định nghĩa” bằng interface**:

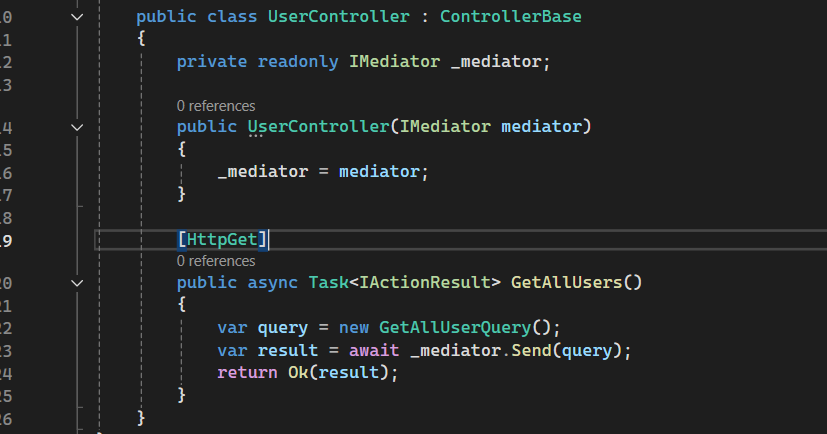
* Data/AppDbContext: Kế thừa DbContext của EF Core — định nghĩa bảng (DbSet), kết nối DB, và ánh xạ Entity (Domain).
* Migrations: Chứa các file migration (EF Core sinh ra) để tạo, cập nhật schema DB.
* Repositories/UserRepository: Là nơi implement lại IRepository (đã định nghĩa ở Application).
* InfrastructureDI: Chỗ đăng ký Dependency Injection cho toàn tầng Infrustructure.

**BƯỚC 4 — TẦNG API**

**Tầng API không xử lý logic trực tiếp. Thay vào đó, nó chỉ đóng vai trò trung gian nhận request, rồi gửi request đó đến Application thông qua MediatR, và trả lại response cho client.**

Controller → MediatR → Handler (Application) → Repository (Infrastructure).





Luồng chạy làm sao gọi được.

| **Bước** | **Ai tham gia** | **Hành động** | **Ý nghĩa / Ghi chú** |
| --- | --- | --- | --- |
| 1 | Client | Gửi request HTTP → GET /api/user | Yêu cầu dữ liệu từ hệ thống |
| 2 | ASP.NET Core Routing | Tìm controller tương ứng: UserController | Xác định nơi tiếp nhận request |
| 3 | Dependency Injection Container | Khởi tạo UserController, bơm sẵn IMediator vào constructor | Controller không cần biết ai tạo mediator |
| 4 | UserController | Tạo query new GetAllUserQuery() và gửi đi \_mediator.Send(query) | Controller chỉ gửi yêu cầu, không xử lý logic |
| 5 | Mediator | Nhận query, tìm đúng Handler tương ứng trong Application layer | Là người điều phối, biết ai sẽ xử lý yêu cầu |
| 6 | Handler (Application layer) | Gọi xuống Repository để lấy dữ liệu (qua Interface) | Thực thi nghiệp vụ, truy xuất dữ liệu khi cần |
| 7 | Repository (Infrastructure layer) | Lấy dữ liệu từ database, ánh xạ sang Entity hoặc DTO | Làm việc trực tiếp với DB |
| 8 | Handler | Nhận dữ liệu, xử lý xong trả về Mediator | Tổng hợp kết quả nghiệp vụ |
| 9 | Mediator | Trả kết quả cho Controller | Luồng xử lý hoàn tất |
| 10 | Controller | Trả Ok(result) → Response JSON về Client | Kết thúc vòng đời request |