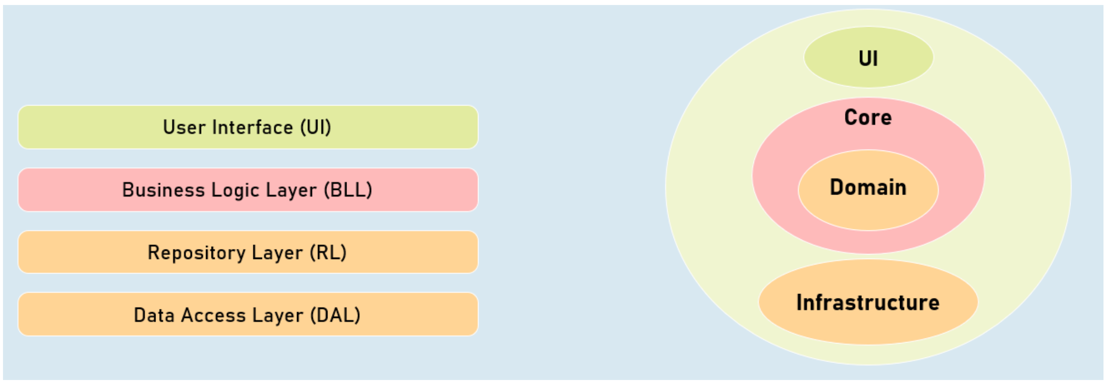
Section Cheat Sheet (PPT)

Overview of Clean Architecture

Instead of "business logic" depend on "data access logic", this dependency is inverted; that means, the "data access logic" depend on "business logic".

Benefit: The business logic is highly clean-separated, independent of data storage and UI, unit-testable.



Traditional Three-Tier / N-Tier Architecture

1. User Interface (UI)
2. Business Logic Layer (BLL)
3. Repository Layer (RL)
4. Data Access Layer (DAL)

Clean Architecture

UI

1. Controllers, Views, View Models

Core

1. Business Logic Services
2. Business Logic Interfaces
3. Data Transfer Objects (DTO)

Domain

1. Repository Interfaces
2. Entity Classes

Infrastructure

1. DbContext, Repositories
2. External API Calls

Clean Architecture

Changing external system

Allows you to change external systems (external APIs / third party services) easily, without affecting the application core.

Scalable

You can easily scale-up or scale-out, without really affecting overall architecture of the application.

Database independent

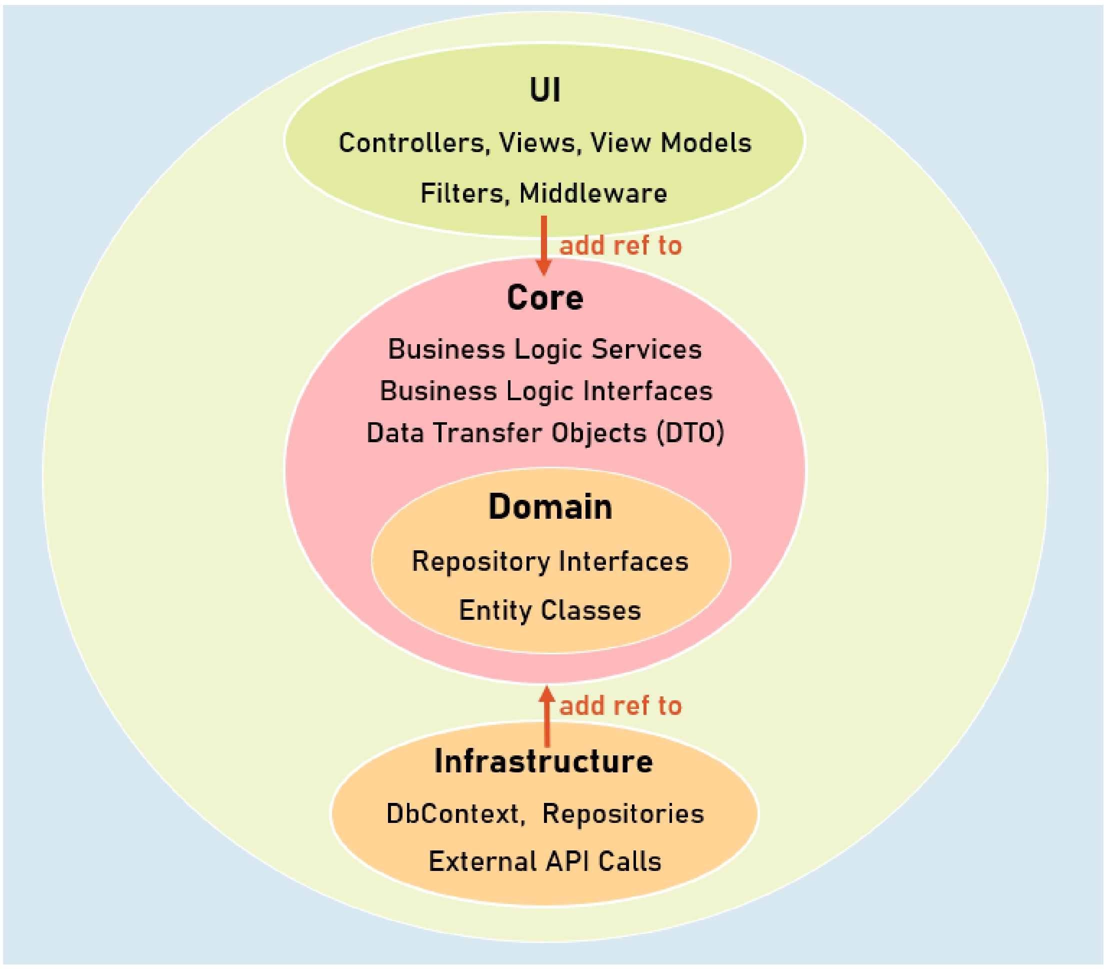
The application core doesn't depend on specific databases; so you can change it any time, without affecting the application core.

Testable

The application core doesn't depend on any other external APIs or repositories; so that you can write unit tests against business logic services easily by mocking essential repositories.

Clean architecture is earlier named as "Hexagonal architecture", "Onion architecture", "Domain-Driven Design", "Vertical Slice Architecture". Over time, it is popular as "clean architecture".

Clean Architecture



UI
Controllers, Views, View Models
Filters, Middleware
add ref to
Core
Business Logic Services
Business Logic Interfaces
Data Transfer Objects (DTO)
Domain
Repository Interfaces
Entity Classes
add ref to
Infrastructure
DbContext, Repositories
External API Calls