Clean Architecture:

* s/w design approach
* Goal: separate concerns so that your code is flexible, testable, and independent of frameworks or databases.
* Core idea
  + Organize code into layers.
  + Dependencies point inward (outer layers depend on inner layers, not the other way around).
  + Business rules (core logic) stay independent from UI, database, or external services.
* Layers of Clean Architecture (Onion-like structure)
  + Entities (core business logic)
  + Use cases (application business logic)
  + Interface adapters (controllers, gateways, presenters)
  + Framework and Drivers (outer layer)

\*\*\*\* Clean Architecture = separating business logic from infrastructure.

* Inner layers = rules (stable).
* Outer layers = details (replaceable).

**Entities (Core):** The rules of the business (what must always be true).

**Use Cases:** The actions/routines of the system

**Interface Adapters:** The translators between inner rules and outside systems.

**Frameworks/Drivers:** The external tools you plug in (DB, UI, libraries).

\*\*\*\* **Dependencies always point inwards** → Core logic doesn’t depend on frameworks, but frameworks depend on core logic.

**Entities** = what your system is (rules).

**Use Cases** = what your system does (actions).

**Adapters** = how your system talks to outside world.

**Frameworks** = tools you plug in.

**Adapters**:

* Think of adapters as connectors or translators between your core business logic and the outside world. Why? Because your core (Entities + Use Cases) doesn’t know about things like databases, HTTP requests, or JSON.
* Types of adapters:
  + Input adapters (Controllers): Bring data from outside into your use cases.
  + Output Adapters (Gateways/Repositories): Send data from use cases to the outside world.
* Why Adapters Exist:
  + To isolate your core business rules from changes in external systems.
  + If tomorrow you switch from PostgreSQL → MongoDB, only the adapter changes, not the use case.
* Adapters are translators that connect your pure business logic to messy external systems (HTTP, DB, queues).

\*\*\*\* Entities vs DTO vs Model flow?????

Key points:

* **Entities** = core rules/data (can exist alone).
* **Use Cases** = orchestrate actions using entities (needs entities to work).
* Direction of dependency: Use Case → Entity, not the other way around.

**Benefits of Clean Architecture** - <https://chatgpt.com/s/t_68b71a2638548191b2cb06bbb4f113d7>

Why Clean Architecture is widely used in banking applications????

* <https://chatgpt.com/s/t_68b71b83da2c8191a61c7fc843bac960>