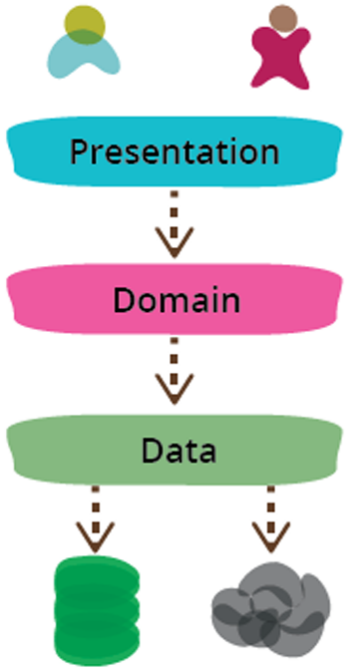
ARCHITECTURE

> In this file, there will be a brief overview of software architecture styles, focusing on monolithic and microservices architectures. The goal is to have the main concepts and only the most important details of the architectures and styles we will use (in BACKEND). The deeper details will be covered in the subject "Information Systems Desing (DSI)".

MONOLITHIC

Layered Architecture Style

- The system is organized into layers, each with a specific responsibility.
- Each layer can only interact with the layer directly below it.
- Common layers include:
  - Presentation Layer (UI)(html, css, js)
  - Business Logic Layer (BLL)(services and controllers)
  - Data Access Layer (DAL)(repositories)
  - Database Layer (DB)(SQL, NoSQL)
- Image of the layered architecture:



MICROSERVICES

```java
// Example of a simple interface in Java
public interface PaymentService {
    void processPayment(double amount);
    boolean refundPayment(String transactionId);
}
```

Understanding an API: Quick Summary

1. What Is an API?

An **API (Application Programming Interface)** is a set of **endpoints** that you expose so other systems can interact with your application.

2. Endpoints and URIs

- **Endpoint = URI + HTTP method** (GET, POST, PUT, DELETE).
- The **URI** points to a resource (e.g., `/users/1`).
- The **HTTP method** defines the action (e.g., GET → fetch, POST → create).
- Your **API = collection of endpoints**.

3. Implementation Details Don't Matter to the Client

- Clients don't care how your code works internally.
- They only care that when they call an endpoint, they get the **expected response**.

4. Documentation and Consistency

- Use documentation tools like **Swagger/OpenAPI** to describe your API.
- The goal is to ensure others know:
  - Which endpoints exist.
  - What input they require.
  - What output they return.
- The key is **reliability and consistency**.

5. Interfaces Inside a Project

- Similarly, an **interface** in your code is like a contract between classes.
- It defines what methods are available without specifying the implementation.
- Just like an API lets external systems know what they can do, an interface lets different parts of your code interact cleanly.

☑ Takeaway

An **API** is just a **contract of endpoints**.
Clients don't need to know your implementation, only that those endpoints behave **as documented**.