**Clean Architecture**

Clean Architecture es un enfoque de diseño de software que busca crear sistemas modulares, mantenibles y escalables. Fue propuesto por Robert C. Martin ("Uncle Bob") y se basa en la organización de las dependencias para maximizar la reutilización de código y minimizar el acoplamiento entre módulos.

* **Principios Fundamentales:**
* **Independencia de la infraestructura:** el código de negocio no debe depender de frameworks, base de datos ni interfaces de usuarios.
* **Independencia de la UI:** la interfaz gráfica puede cambiar sin afectar la lógica del negocio.
* **Testabilidad:**  los componentes del sistema deben ser fáciles de probar de manera aislada.
* **Independencia de la base de datos:** se debe poder cambiar la base de datos sin afectar la lógica de la aplicación.
* **Independencia de agentes externos:**  se debe poder modificar bibliotecas externas sin afectar el dominio.
* **Estructura de Clean Architecture:**

CA se representa generalmente con una serie de capas concéntricas, donde las dependencias fluyen desde el exterior hacia el interior:

1. **Entities:**

* Representa las reglas del negocio más generales.
* Son reutilizables en cualquier contexto de aplicación.

1. **Use Cases:**

* Contienen la lógica específica de la aplicación.
* Coordinan el flujo de datos entre entidades y otras capas.

1. **Interface Adapters:**

* Contienen implementaciones de repositorios, controladores, presentadores y gateways.
* Transforman datos para que sean entendidos por otras capas.

1. **Infrastructure ( Frameworks y Drivers)**

* Contiene bases de datos, herramientas de UI y otros detalles externos.
* Es la capa más externa y de menor relevancia en la lógica central.
* **Beneficios de Clean Architecture:**
* Es fácil de mantener ya que se pueden cambiar partes del sistema sin afectar a otras.
* La separación en capas permite una alta reutilización de código.
* Al tener un acoplamiento reducido las pruebas unitarias son más sencillas ya que se ejecutan de manera aislada.
* Se tiene menor dependencia de tecnologías específicas.
* **Ejemplo implementado en el Proyecto:**

El proyecto sigue la estructura mencionada con las siguientes capas:

📂 FP.CleanArchitecture

├── 📂 FP.API Capa de Presentación (Controllers, Endpoints)

├── 📂 FP.Application Capa de Aplicación (Servicios y lógica de negocio)

├── 📂 FP.Domain Capa de Dominio (Entidades y contratos)

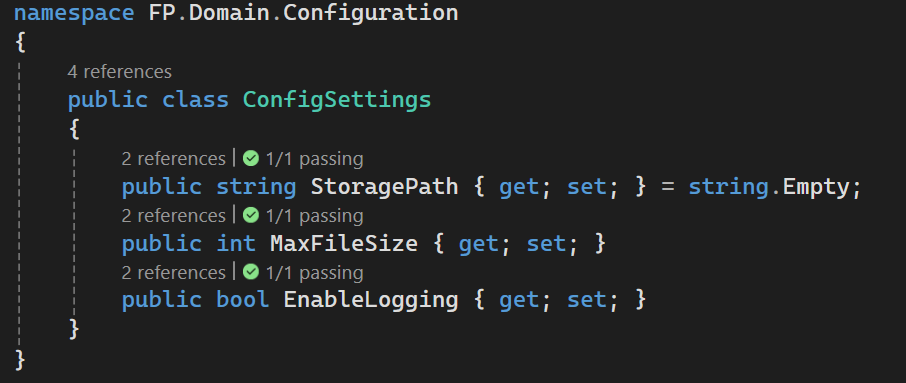
├── 📂 FP.Infrastructure Capa de Infraestructura (Acceso a datos,

integraciones externas)

├── 📂 FP.Tests Pruebas Unitarias e Integración

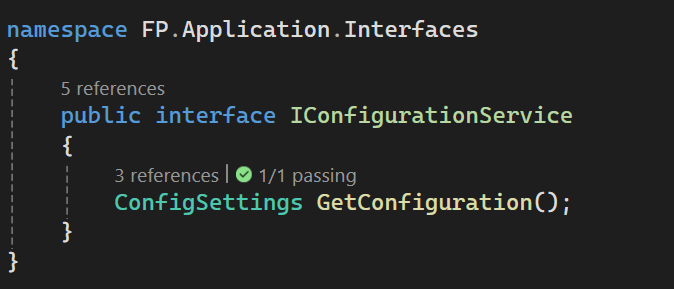
En este caso cada capa tiene una responsabilidad única y está diseñada para ser desacoplada del resto.

**Capa de Dominio:** define las entidades principales y contratos del sistema.



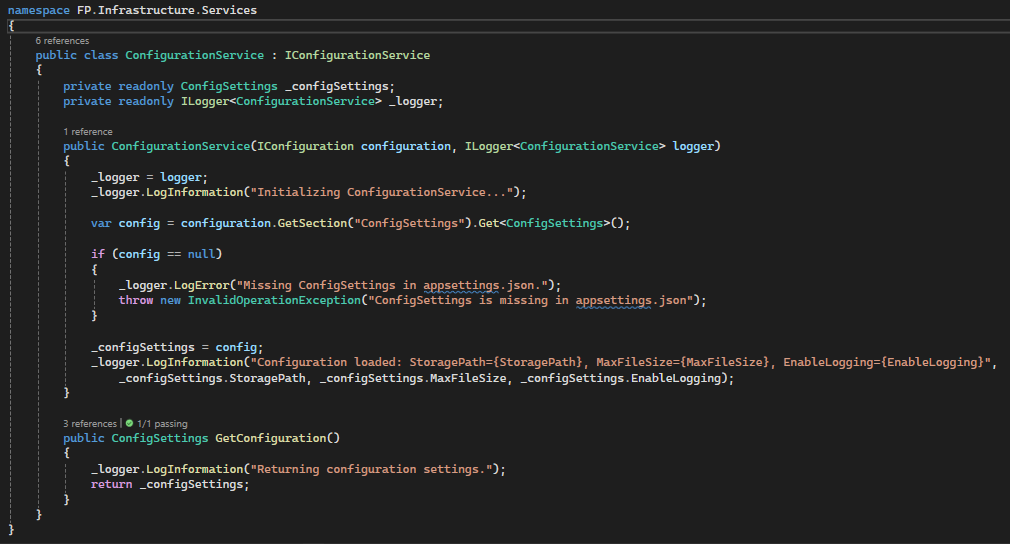
Esta clase representa la configuración del sistema sin depender de ninguna tecnología específica.

**Capa de aplicación:** define la lógica del negocio a través de interfaces.



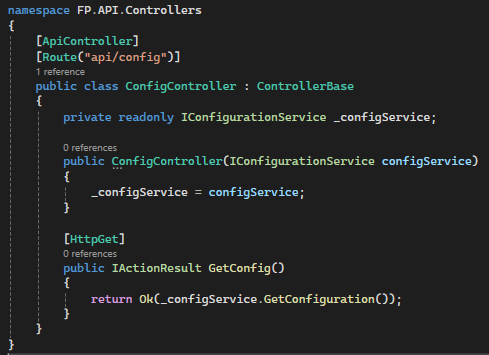
Esta interfaz define el contrato que debe seguir cualquier implementación.

**Capa de infraestructura:** implementa los servicios definidos en la capa de aplicación, utilizando frameworks y tecnologías externas.



Aquí aplicamos Clean Architecture asegurando que la implementación esté aislada del contrato.

**Capa de presentación:** expone los servicios a través de controladores REST.



El controlador solo depende de la interfaz, lo que permite desacoplarlo de la implementación.

**Conclusión:**

Clean Architecture proporciona una estructura flexible y escalable para el desarrollo de software. Separando las responsabilidades en capas, se logra un código más mantenible, reusable y desacoplado de frameworks o tecnologías específicas. Aplicar estos principios puede mejorar la calidad del software y facilitar su evolución a lo largo del tiempo.