UCR | ECCI | CI-0156- Práctica Supervisada
Tarea - Clean Architecture
Fabián Orozco Chaves — B95690
Septiembre, 2023

Tabla de Contenidos

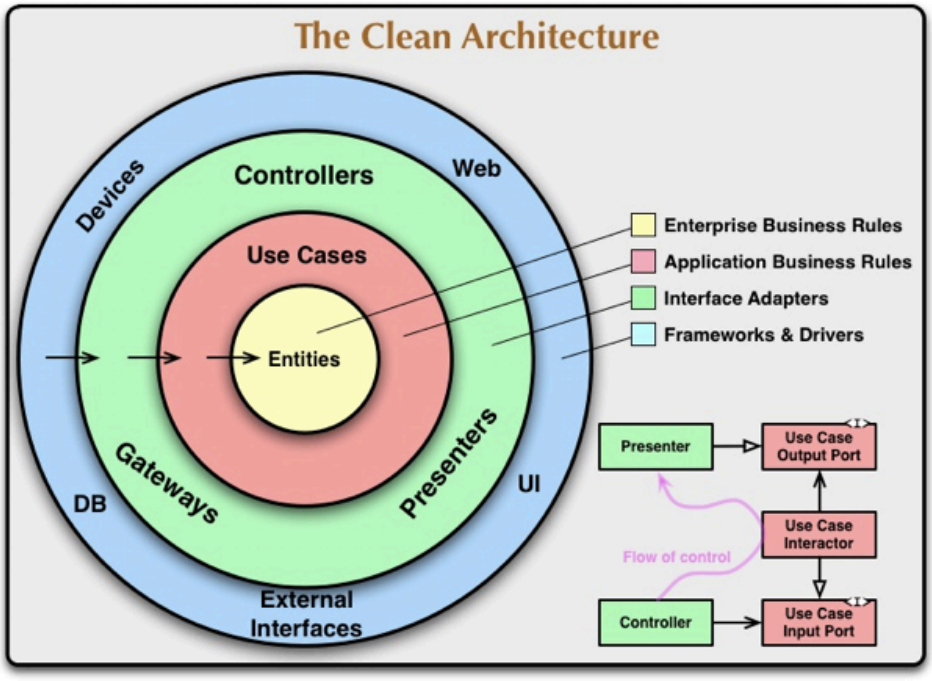
Enunciado

Explique qué es Clean Architecture, ventajas y desventajas.
Además, un incluya un ejemplo de la arquitectura.

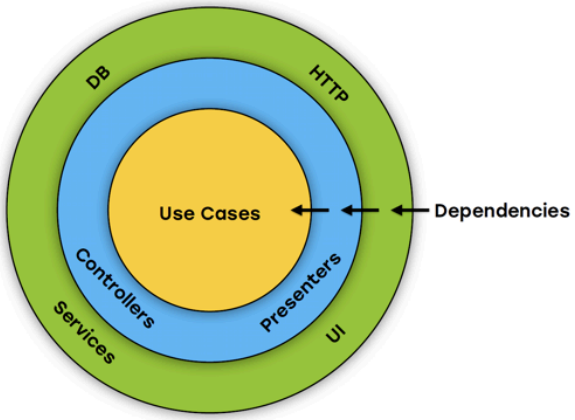
Descripción

La arquitectura en términos de software, representa las decisiones de diseño que dan forma a un sistema, donde lo significativo se mide por el costo del cambio (Martin, 2017).

Clean architecture se centra en separar mediante capas que ejercen una responsabilidad única y con directrices de interacción, con el objetivo de lograr un código más mantenible, escalable y comprobable.



Capas	
Entities (Domain)	Las "Entidades" encapsulan reglas de negocio empresariales, pueden ser objetos con métodos o estructuras de datos y funciones, utilizadas por varias aplicaciones en la empresa. Las reglas de negocio se pueden probar sin necesidad de componentes externos. Ejemplos: entidades, objetos de valor y servicios de dominio.
Use cases (Application)	Los "Casos de Uso" capturan las reglas de negocio de la aplicación, enfocándose en lo que hace la aplicación, no en cómo lo hace, y se centran en reglas específicas de la aplicación. Ejemplos: servicios de aplicación, DTO (objetos de transferencia de datos) y mapeadores.
Interface adapters	Los "Adaptadores de Interfaz" convierten datos desde el formato más conveniente para los casos de uso y entidades, hacia el formato más adecuado para agentes externos como la base de datos o la web. En una arquitectura MVC, esto involucra Presenters, Vistas y Controladores.
Frameworks & drivers (Infrastructure o Presentation)	Los "Frameworks y Controladores" son herramientas como bases de datos y marcos de desarrollo web que proporcionan funcionalidad esencial para la aplicación. Estos componentes actúan como enlaces entre la arquitectura interna de la aplicación y los aspectos externos, como la base de datos y la interfaz web, sin contaminar la lógica central de la aplicación. Ejemplo: implementación del acceso a los datos, registro, correo electrónico y otros mecanismos de comunicación.



Las dependencias en el código fuente sólo pueden apuntar hacia el interior de la aplicación, evitando que los componentes internos dependan de componentes externos. No tienen que ser siempre necesariamente 4 **círculos**. Algunas **otras** arquitecturas orientadas a la separación de preocupaciones y arquitectura limpia son: Hexagonal Architecture (Ports and Adapters), Onion Architecture, Screaming Architecture, DCI (Data, Context, Interaction), BCE (Boundary-Control-Entity)

Clean architecture puede adaptar una serie de principios de ingeniería de software **SOLID y otros**	
Single Responsibility Principle	Cada módulo realiza una única tarea específica, evitando responsabilidades múltiples.

Open Close Principle	El software es extensible sin modificar código existente, fomentando adaptabilidad.
Liskov Substitution Principle	Las subclases pueden reemplazar a sus clases base sin problemas.
Interface Segregation Principle	Interfaces específicas para cada cliente, evitando métodos innecesarios.
Dependency inversion Principle	Los módulos deben depender de abstracciones en lugar de detalles concretos, fomentando flexibilidad.
Principles of Components Cohesion	Los componentes deben tener funcionalidades relacionadas, mejorando comprensión.
Stable & Abstract Dependency Principles	Dependencias estables y abstractas para reducir el acoplamiento.
Dependency Rule	Dependencias apuntan hacia abstracciones, minimizando acoplamiento a detalles concretos.
Services Architecture	Componentes basados en servicios para facilitar modularidad y reutilización.
Design & Code Organization	Estructura y organización lógica que mejora la mantenibilidad y escalabilidad.

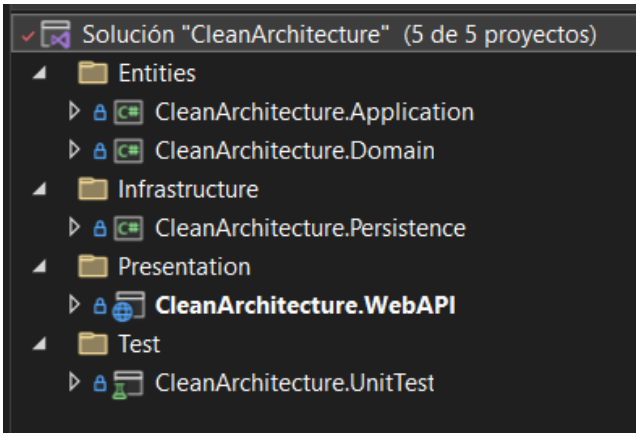
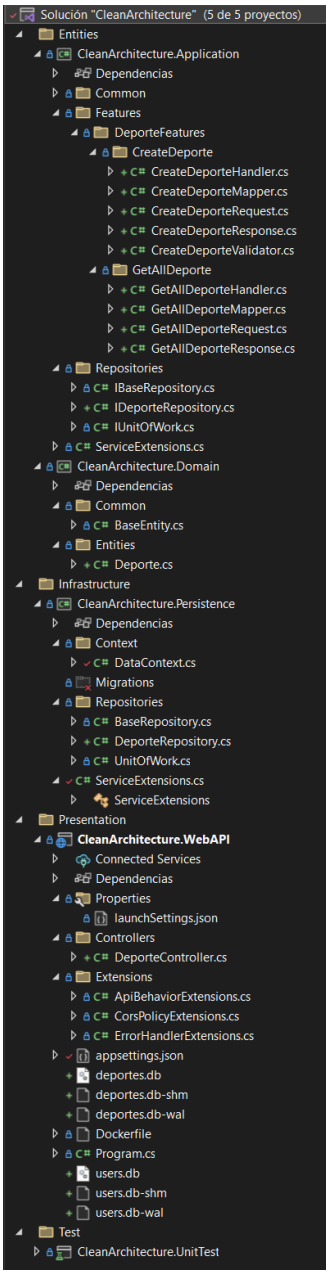
Ventajas

- Facilita la comprensión de la estructura general del código.
- Facilita la modificación o ampliación del sistema en una capa específica sin introducir efectos secundarios en otras capas.
- Permite modificar partes específicas sin afectar otras; facilita las pruebas independientes y reemplazar componentes sin problemas.
- Adaptable en diversos contextos de desarrollo como aplicaciones web, móviles o embebidas.

Desventajas

- Puede ser complicado dividir responsabilidades entre capas (curva de aprendizaje) y es excesiva en proyectos pequeños.
- Incrementa la complejidad de identificar cuellos de botella en el rendimiento.
- Puede provocar problemas de rendimiento por la sobrecarga de llamadas a funciones, creación de objetos y asignación de memoria.
- La transformación de datos (casting) entre capas puede aumentar el uso de la memoria “innecesariamente” y disminuir el rendimiento.

Ejemplo
(adjunto zip)

Estructura simplificada	Estructura completa
	

Descripción del API

Entities (domain)

Contiene **Common**: Esta carpeta almacena la clase BaseEntity y otros agregados. BaseEntity es una clase abstracta con propiedades como Id, DateCreated, DateUpdated y DateDeleted.

Entities: Aquí se guardan todas las entidades de dominio. Por ejemplo, la clase User hereda de BaseEntity y define propiedades como Nombre y JugadoresPorEquipo.

Application

Repositories contiene interfaces de repositorio que definen métodos para **acceder a los datos**, como lectura, creación, actualización y eliminación, y también incluye la interfaz de unidad de trabajo que guarda cambios.

Features tiene características funcionales (como crear), cada característica tiene su propia subcarpeta y contiene handler, request, response, DTOs, mapeadores, validadores, utilizan bibliotecas como **MediatR** para la gestión de comunicación, **AutoMapper** para la conversión de objetos y **FluentValidation** para validación.

Common almacena contenido común como helpers, excepciones y comportamientos compartidos.

ServiceExtensions contiene un método de extensión para configurar la inyección de dependencias en la capa de aplicación.

Infrastructure

Contiene una biblioteca de clases llamada **Persistence** que contiene la implementación de las interfaces de repositorio. Hace referencia a la capa de aplicación.
Context: alberga la clase de contexto de Entity Framework (u otro ORM).

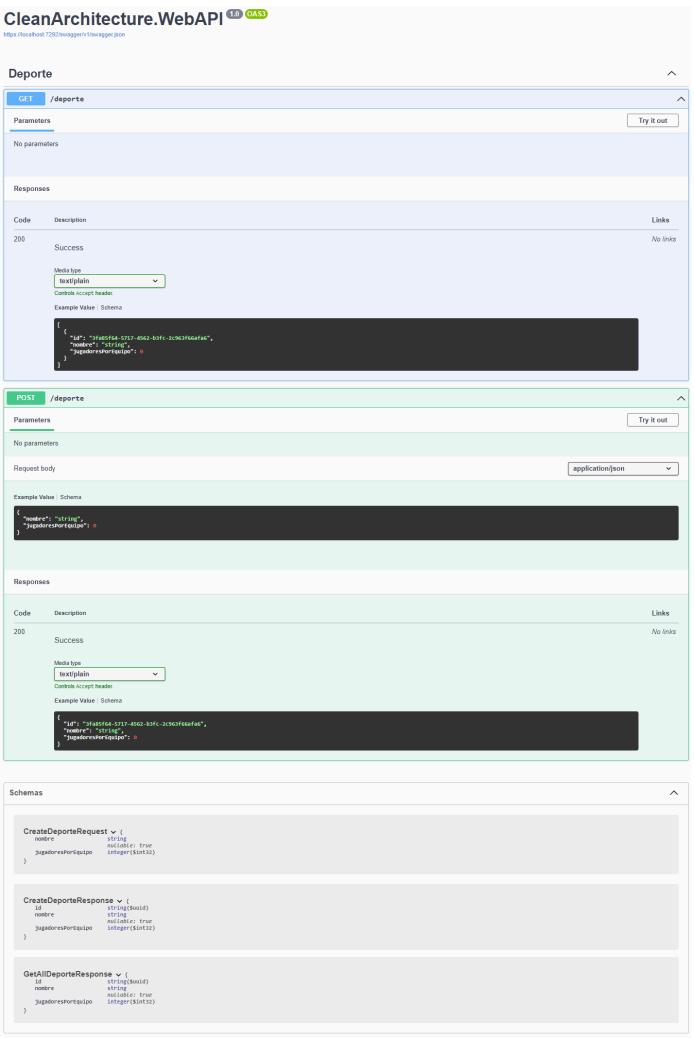
Repositories: contiene la implementación de las interfaces de repositorio y unidad de trabajo.

ServiceExtensions: para configurar la inyección de dependencias en la capa de infraestructura.

Presentation (proyecto WebAPI)

Controllers: maneja las solicitudes HTTP entrantes y devuelve la respuesta HTTP adecuada. Actúa como intermediario entre el cliente y la lógica del servidor.

Extensions: La carpeta "Extensions" contiene clases y configuraciones de métodos de extensión para el proyecto Web API, como el manejador de errores global, la política CORS, comportamientos, etc.



El resultado del **API** muestra información sobre solicitudes **GET** y **POST** que si hicieron sobre la entidad **Deporte** recibiendo un **200 OK** como respuesta, además de tres **esquemas** de **solicitud** y **respuesta** para **crear** y **obtener** deportes.

Referencias Bibliográfica

1. (Descripción) CLEAN ARCHITECTURE
2. (Descripción) The Clean Code Blog by Robert C. Martin (Uncle Bob)
3. (Descripción, ventajas y desventajas) Clean Architecture
4. (Más ventajas) CLEAN ARCHITECTURE LAYERS – WHAT THEY ARE AND THE BENEFITS.
5. (Más desventajas) Performance Disadvantages of Clean Architecture: A Closer Look.
6. (Ejemplo) Clean Architecture in ASP .NET Core Web API