Curso .NET Core RESTful API

Inversion de Control

- Que es el Principio de Inversion de Control? Se refiere a todo aquel diseño de software cuyo proposito obedece a la necesidad de querer controlar el flujo de ejecucion de este, de forma automatica y transparente, es decir, ceder el control de ese flujo a un "agente externo", normalmente un framework.
- ***Llamando tambien IoC***

Inyeccion de Dependencieas

- Que es el principio de inyeccion de dependencias?
- Es un patron de diseño que sirve para "inyectar" componentes a las clases que tenemos implementadas. Esos componentes son contratos que necesitan nuestras clases para poder funcionar, de ahi el concepto de "dependencias".
- La diferencia sustancial en este patron de diseño es que nuestras clases no crearan esos objetos que necesitan sino que se les suminitrara otra clase "contenedora" perteneciente al Framework DI que estomos utilizando y que inyectaran la implementacion deseada a nuestro contrato, y todo ello si tener que hacer un solo "new"
- Para resolver ese problema normalmente los frameworks DI utilizan en la inyenccion de dependencias interfaces entre componentes.
- ***Onion Architecture***
- ***Jeffrey Palermo*** Introdujo esta arquitectura

Que es Onion Architecture?

- La mayoria de la arquitectura tradicionales plantean problemas fundamentales de acoplamiento estrecho y separacion de preocupaciones.
- Jeffrey Palermo introdujo Onion Architecture para Proporcionar una mejor manera de crear aplicaciones en la perspectiva de una mejor capacidad de prueba mantenibilidad y confiabilidad.
- Onion Architecture aborda los desafios que enfrentan las arquitectura de 3 y n niveles, y proporciona una solucion para problemas comunes. las capas de la arquitectura Onion interactuan entre si mediante el uso de interfaces.
- ***El principio de Onion Architecture***
- Onion Architecture se basa en la inversion del principio de control. Onion Architecture se compone de multiples capas concentricas que se interconectan entre si hacia el nucleo que representa el dominio.
- La arquitectura no depente de la capa de datos como en las arquitectura clasicas de varios niveles, sino de los modelos de dominio reales.

Architecture de 3 Capas

- En arquitectura de 3 y n niveles, ninguna de las capas es independiente; este hecho suscita una separacion de proocupaciones. Estos sistemas son muy dificiles de entender y mantener. El incoveniente de esta arquitectura tradicional es el acoplamiento innecesario.

1. Presentation Layer
2. Business Layer
3. Data Access Layer

- Data Source

El objectivo principal es controlar el acoplamiento

- Onion Architecture resolvio estos problemas definiendo capas desde el nucleo hasta la infraestructura. Aplica la regla fundamental moviendo todos los acoplamientos hacia el centro. Esta arquitectura esta indudablemente sesgada hacia la programacion orientada a objetos y antepone los objetos los demas.

User Interface | Application Service | Domain Service | Domain Model | Infrastructure | Tests

Capa de Dominio

- Este es el centro de nuestra arquitectura. Contiene todas las entidades del dominio. Estas entidades de dominio no tienen ningun tipo de dependencias. Son planos sin una logica de codigo masiva o compleja.

Capa Servicios de dominio

- Esta parte suele ser la encargado de definir las interfaces necesarias para permitir almacenar y recuperar objetos. Si lo prefiere, podemos convertir esta definicion en una definicion de patron de repositorio.
- Sin embargo, este tipo de comportatimeinto tiene su implementacion en la capa mas alejada del centro, nombrado por infraestructura.

Capa Servicios de Aplicacion

- Esta capa tiene el proposito de abrir la puerta al corazon de "la cebolla". El sistema central esta compuesto por tres aros de "cebolla" Modelo de dominio, Servicios de dominio y el actual. Contamos con las implementaciones de las interfaces core para porder brindar estas funcionalidades al anillo externo (UI,

Infraestructura y Tests).

Capa UI, Infraestructura y pruebas

- El "aro de cebolla" exterior es para componentes que cambian con frecuencia. Aqui es donde vive la interfaz de usuario con todas sus vistas, controladores y todo tipo de cosas relacionadas con la logica de presentacion. Este anillo tiene una implementacion para el principio de inyeccion de dependencia.

CQRS Pattern

- De que trata este patron?
- El patron de segregacion de responsabilidad de consultas y comandos (CQRS) separa las operaciones de lectura y actualizacion de un almacen de datos.
- La implementacion de CQRS en su aplicacion puede maximizar su rendimiento, escalabilidad y seguridad. La flexibilidad creada por la migracion a CQRS permite que un sistema evolucione mejor con el tiempo y evita que los comandos de actualizacion causen conflictos de fusion a nivel de dominio.
- ***El enfoque principal es sperar la forma de leer y escribir datos.***

El problema que solventa este patron

- En las arquitectura tradicionales, se utiliza el mismo modelo de datos para consultar y actualizar una base de datos. Eso es simple y funiona bien para operaciones CRUD basicas.
- En aplicaciones mas complejas, sin embargo, este enfoque puede volverse dificil de manejar. Por ejemplo, en el lado de la lectura, la aplicacion puede realizar muchas consultas diferentes, devolviendo objetos de transferencia de datos (DTO) con diferentes formas.
- El mapeo de objetos puede volverse complicado. En el lado de la escritura, el modelo puede implemtar una validacion compleja y logica empresarial. Como resultado, puede terminar con un modelo demasiado complejo que hace demasiado.