ADR002 - SIGEA

Developement Team

Indice

ADR 002: [Modificacion de Arquitectura de Microservicios Orientado a Eventos por Arquitectura en Capas + Clean Architecture + MVC SIGEA]

Estado

[Aceptado]

Contexto

Se esta desarrollando un sistema que digitaliza y automatiza la gestión integral de actividades de extensión universitaria como cursos, talleres, conferencias y diplomados.

Se requiere de una arquitectura que permita que el sistema escale me manera rapida, que facilite las modificacionesy y que sea sencilla de desarrollar e implementar para los integrantes del equipo de desarrollada.

Decision

Se decidio modificar la arquitectura planteada de `Microservicios Orientado a Eventos` por una Clean Architectura junto con el patron de diseño MVC.

El motivo principal de este cambio fue debido a que el sistema no necesita la escalabilidad que proporciona `Microservicios`, ademas de que los eventos pueden ser manejados sin una `Arquitectura Orientada a Eventos`. Otro motivo clave para la decision es la alta deuda tecnica que iba a conllevar implementar estas arquitecturas, debido a que el equipo no se encuentra relacionada con estas y se necesita un sistema completo en un tiempo de 2 meses.

Las capas se dividiran de la siguiente manera:

```
[ UI (Capa de presentación) ]
   Controlador (C) - Recibe requests del usuario
   Modelo (M) - Datos que se muestran (DTOs, ViewModels)

[ Capa de aplicación / casos de uso ]
   Orquesta la lógica de negocio
   Llama a servicios del dominio

[ Capa de dominio ]
   Entidades, lógica de negocio pura
   Interfaces (repositorios, servicios)

[ Capa de infraestructura ]
   Implementaciones reales: bases de datos, APIs externas, etc.
```

Capa capa haria lo siguiente:

1. *Capa de Presentacion*: Muestra datos al usuario y recibe sus acciones del usuario. No tiene lógica de negocio.
2. *Capa de Aplicacion*: Orquesta casos de uso y no tiene lógica de negocio compleja, ademas de que usa interfaces del dominio.
3. *Capa de Dominio*: Contiene el "corazón" de la aplicación y no depende de ninguna tecnología. Contiene lógica pura y reglas de negocio. Tambien contiene abstracciones e interfaces que establecen contratos con otras capas.
4. *Capa de Infraestructura*: Implementa repositorios, conectores, clientes externos. Puede depender de ORMs, SDKs, etc.

La interaccion entre las capas seria la siguiente:

Capa	Conoce a	Tipo de Dependencia	Ejemplo
Presentación	Aplicación	directa	Controlador llama caso de uso
Aplicación	Dominio (interfaces)	directa	UseCase usa repositorios
Dominio	Nadie	—	Core puro
Infraestructura	Dominio (interfaces)	implementación inversa	Implementa interfaces de core

Consecuencias

Positivas

- *Mantenibilidad*: El patron Clean Architecture permite una organizacion desacoplada y con alta coherencia, lo que facilita la mantenibilidad a futuro.
- *Desarrollo e Implementacion más Sencilla y Rapida*: El equipo esta familiarizado con la arquitectura en capas y el patron de diseño MVC, por lo que el tiempo para comprender el flujo de trabajo se reduce.

Negativas

- *Baja Escalabilidad*: La arquitecutra en capas no es muy conocida por permitir la escalabilidad horizontal.
- *Baja Flexibilidad*: Nuevos requisitos o funcionalidades podrian necesitar la adicion de nuevas capas, lo cual aumentaria la complejidad del esquema actual de 4 capas.
- *Mayor Probabilidad de Errores con el Control de Versiones*: Esta arquitectura no permite que cada integrante trabaje de manera completamente independiente en su area o modulo, lo cual podria traer problemas en un futuro al momento de juntar los cambios.