**Documento de Arquitectura - API Marvel**

**1. Introducción**

Este documento describe el patrón de arquitectura utilizado en el desarrollo de la API Marvel, proporcionando detalles sobre la estructura del código, los principios de diseño aplicados y las mejores prácticas adoptadas.

**2. Patrón de Arquitectura**

La API Marvel sigue el patrón de arquitectura **Clean Architecture**, patrón promueve la separación de responsabilidades y facilita la escalabilidad y mantenibilidad del código.

**2.1 Capas de la Arquitectura**

La aplicación está dividida en varias capas lógicas, cada una con una responsabilidad bien definida:

1. **Capa de Presentación (API/Controllers)**
   * Exponen los endpoints REST utilizando **ASP.NET Core Web API**.
   * Reciben y validan las solicitudes HTTP.
   * Delegan la lógica de negocio a la capa de aplicación.
2. **Capa de Aplicación (Application Services)**
   * Contiene la lógica de negocio y reglas de aplicación.
   * Implementa casos de uso y servicios.
   * Define interfaces para la comunicación con la capa de infraestructura.
3. **Capa de Dominio (Domain Models)**
   * Define las entidades del dominio y sus reglas de negocio.
   * Se encuentra completamente desacoplada de frameworks y librerías externas.
4. **Capa de Infraestructura (Persistence & External Services)**
   * Implementa la persistencia de datos utilizando **Entity Framework Core**.
   * Maneja la integración con la API de Marvel.
   * Contiene la configuración de acceso a bases de datos y servicios externos.

**3. Tecnologías Utilizadas**

Para la implementación de esta arquitectura, se han utilizado las siguientes tecnologías:

* **ASP.NET Core Web API**: Para la exposición de los servicios REST.
* **Entity Framework Core**: Para la gestión de la base de datos y migraciones.
* **SQL Server**: Como motor de base de datos.
* **Swagger**: Para documentar y probar la API.
* **Inyección de Dependencias**: Para la gestión de servicios desacoplados.

**4. Patrón de Diseño**

* **Repository Pattern**: Para la abstracción de la lógica de acceso a datos.
* **Dependency Injection**: Para desacoplar la implementación de las dependencias.
* **DTOs (Data Transfer Objects)**: Para estructurar y transferir datos entre capas.

**5. Flujo de Datos**

1. Un cliente realiza una petición HTTP a un endpoint de la API.
2. El **Controller** recibe la solicitud y la delega a un servicio de aplicación.
3. El **Servicio de Aplicación** ejecuta la lógica de negocio y consulta la capa de infraestructura si es necesario.
4. La **Capa de Infraestructura** obtiene o almacena datos en la base de datos o en la API externa de Marvel.
5. Se retorna una respuesta estructurada al cliente.

**6. Conclusión**

La implementación de **Clean Architecture** en la API Marvel permite una organización modular y escalable, facilitando futuras mejoras y mantenimientos. Además, el desacoplamiento entre capas permite realizar cambios sin afectar otras partes del sistema.