OpenPeer — Aprendizaje Paso a Paso (Sesiones 1 y 2)

Este documento describe el trabajo realizado en las **Sesiones 1 y 2** del proyecto OpenPeer. El objetivo es que entiendas no solo el *qué* implementamos, sino también el *cómo* y *por qué* de las tecnologías usadas (.NET, EF Core, Docker, CQRS, MediatR, FluentValidation). ---

Sesión 1 — Bootstrap del Proyecto

Objetivo: Crear la base del proyecto siguiendo la filosofía de Clean Architecture, y levantar una base de datos SQL Server en Docker lista para usar con EF Core.

Implementación técnica:

- Se creó la solución con cuatro proyectos: Domain, Application, Infrastructure y Api.
- Se configuró Entity Framework Core en Infrastructure con un DbContext.
- Se generó una migración inicial y se aplicó a la BD.
- Se preparó un contenedor de SQL Server con Docker y un docker-compose.yml.
- Se añadió un endpoint de /health en la API.
- Todo se versionó en Git y se documentó en README.md.

Conceptos y tecnologías:

- **Entity Framework Core (EF Core):** Un ORM (Object-Relational Mapper) que permite mapear clases C# a tablas SQL. Facilita trabajar con LINQ en lugar de escribir SQL manual. - **DbContext:** La clase principal de EF Core que gestiona la conexión a la BD y define conjuntos de entidades (`DbSet`). - **Docker:** Tecnología que permite ejecutar software en contenedores aislados. Aquí se usa para levantar SQL Server sin necesidad de instalarlo directamente. - **docker-compose:** Herramienta para definir y ejecutar varios contenedores con un archivo YAML. Definimos un servicio `sqlserver` con credenciales, volumen y healthcheck.

Sesión 2 — Articles con CQRS y Validación

Objetivo: Añadir endpoints reales de negocio para `Articles`, aplicando buenas prácticas modernas (CQRS, MediatR y FluentValidation).

Implementación técnica:

- Se añadió MediatR v13 y se configuró en Program.cs para descubrir automáticamente los handlers.
- Se incorporó FluentValidation para validar entradas de usuario.
- Se creó la estructura Application/Articles con carpetas Commands, Queries, Validators y Dtos.
- Se implementó CreateArticle (Command) con su validador.
- Se implementaron GetArticleById y GetArticlesPaged (Queries).
- Se añadieron endpoints en Minimal API: POST /articles, GET /articles/{id}, GET /articles.
- Se probó todo en Swagger con ejemplos de request/response.

Conceptos y tecnologías:

- **CQRS (Command Query Responsibility Segregation):** Patrón que separa claramente las operaciones de escritura (Commands) de las de lectura (Queries). Esto aporta claridad, mejor testabilidad y escalabilidad. - **MediatR:** Implementa el patrón Mediator. En lugar de que los controladores llamen a servicios directamente, envían un mensaje (Command o Query) a través de un mediador, que resuelve y ejecuta el Handler correcto. - **Handlers:** Clases que contienen la lógica para cada Command o Query. Ejemplo: CreateArticleHandler guarda un artículo en la BD. - **FluentValidation:** Librería que permite definir reglas de validación de forma declarativa. Ejemplo: `RuleFor(x => x.Title).NotEmpty().MaximumLength(150);`. - **DTOs (Data Transfer Objects):** Objetos ligeros usados para devolver datos a la API. Evitan exponer directamente las entidades de dominio. - **Dependency Injection (DI):** Principio en el que las dependencias no se crean dentro de las clases, sino que se inyectan por el contenedor de servicios. En .NET, se configura con `builder.Services.Add...` y se resuelve automáticamente.

Conclusión Tras las dos primeras sesiones tienes: - Una solución .NET 8 organizada en capas (Domain, Application, Infrastructure, Api). - SQL Server ejecutándose en Docker y conectado con EF Core. - Una migración inicial aplicada y validada. - Endpoints REST de Articles listos con CQRS, MediatR y FluentValidation. **Valor añadido:** Has aprendido no solo a montar un proyecto funcional, sino también a usar tecnologías modernas que aparecen en entrevistas técnicas y en empresas reales: Entity Framework Core, Docker, CQRS, MediatR, FluentValidation y DI. **Siguiente paso (Sesión 3):** Incorporar Reviews y EditorialDecisions, además de manejar concurrencia optimista con RowVersion.