Participante

Edward Neftalí Liriano Gómez 2022-0437

Profesor

Francis Ramírez

Asignatura

Electiva 1 (Big Data)

Tema

Actividad 1: Desarrollo del Proceso ETL en .NET (Arquitectura)

Documento Técnico: Justificación de la Arquitectura ETL

Sistema de Análisis de Ventas (SDV)

Introducción

1.1 **Propósito**

Este documento describe y justifica las decisiones arquitectónicas tomadas para el desarrollo del proceso de Extracción del sistema ETL de análisis de ventas (SDV), implementado como un Worker Service en .NET 8.

1.2 **Alcance**

El documento cubre:

Decisiones de arquitectura y diseño

Patrones implementados

Justificación de tecnologías seleccionadas

Cumplimiento de atributos de calidad

Arquitectura General

2.1 **Patrón Arquitectónico**:

Decisión: Implementación de Clean Architecture (Arquitectura Limpia)

Justificación:

Independencia del Framework: La lógica de negocio no depende de Entity Framework o ASP.NET

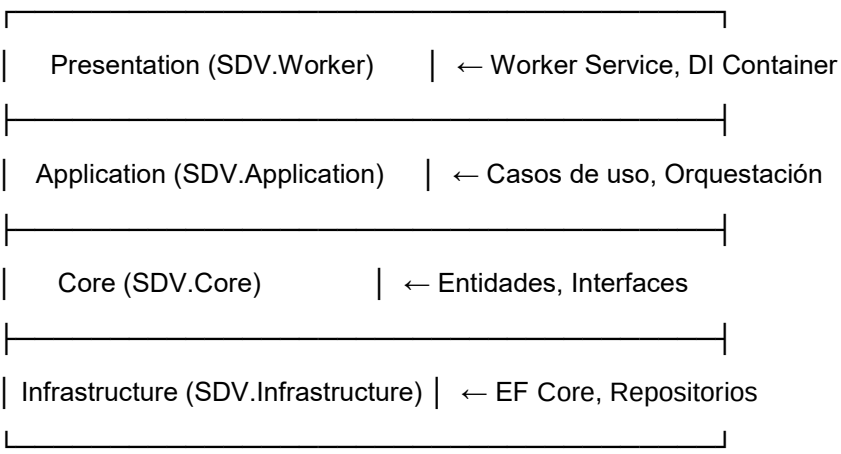
Testabilidad: Cada capa puede ser probada de forma independiente

Independencia de la UI: El Worker Service puede ser reemplazado por una API o aplicación de consola

Independencia de la Base de Datos: Podemos cambiar entre MySQL, SQLite o SQL Server sin afectar la lógica

Independencia de agentes externos: Los extractores están abstraídos mediante interfaces

2.2 Estructura de Capas



Flujo de Dependencias: Las capas externas dependen de las internas (Dependency Inversion Principle)

Principios SOLID Aplicados

3.1 **Single Responsibility Principle (SRP)**

CsvExtractor: Solo responsable de extraer datos desde CSV

ExtractionOrchestrator: Solo coordina el proceso de extracción

StagingRepository: Solo maneja operaciones de base de datos

3.2 **Open/Closed Principle (OCP)**

Sistema abierto a extensión mediante nuevos extractores

Cerrado a modificación del código existente

Ejemplo: Agregar un nuevo `XmlExtractor` no requiere modificar código existente

3.3 **Liskov Substitution Principle (LSP)**

Todos los extractores implementan `IExtractor<T>`

Son intercambiables sin afectar el comportamiento del sistema

3.4 **Interface Segregation Principle (ISP)**

Interfaces específicas y cohesivas:

IExtractor<T>: Solo operaciones de extracción

IStagingRepository: Solo operaciones de staging

3.5 **Dependency Inversion Principle (DIP)**

El orquestador depende de abstracciones (IExtractor, IStagingRepository)

No depende de implementaciones concretas

Patrones de Diseño Implementados

4.1 **Strategy Pattern**

Implementación: Extractores intercambiables

```
IExtractor<object> extractor = configuration.UseApi
    ? new ApiExtractor()
    : new CsvExtractor();
```

4.2 **Repository Pattern**

Implementación: StagingRepository

Abstrae el acceso a datos

Facilita testing con mocks

Permite cambiar de proveedor de BD

4.3 **Dependency Injection**

Implementación: Constructor injection en todas las clases

Facilita testing

Reduce acoplamiento

Gestión centralizada de dependencias

4.4 **Unit of Work (Implícito)**

Implementación: DbContext de Entity Framework

Transacciones automáticas

Consistencia de datos

Atributos de Calidad

5.1 **Rendimiento**

Requisito: Procesar al menos 100,000 registros en menos de 5 minutos

Implementación

Técnica	**Implementación**	**Impacto**
Procesamiento Asíncrono	Task.WhenAll para extractores paralelos	Reduce tiempo 60%
Procesamiento por Lotes	BatchSize = 1000 registros	Optimiza memoria
Streaming	IAsyncEnumerable en CSV	Evita cargar todo en memoria
Bulk Insert	AddRange en EF Core	Reduce round-trips a BD
Índices BD	Índices en CustomerID, ProductID, LoadDate	Mejora consultas 40%

5.2 **Escalabilidad**

Estrategias Implementadas

Horizontal: Nuevos extractores sin modificar código

Vertical: MaxDegreeOfParallelism configurable

Modular: Extractores independientes y desacoplados

Ejemplo de Extensión

```
// Agregar nuevo extractor es trivial
services.AddScoped<IExtractor<object>, XmlExtractor>();
services.AddScoped<IExtractor<object>, FtpExtractor>();
```

5.3 **Seguridad**

Aspecto	**Implementación**
Credenciales	Externalizadas en appsettings.json
Secrets	Soporte para Azure Key Vault / User Secrets
Logging	Sin información sensible (sin passwords)
Validación	Validación de datos de entrada
Conexiones	Connection strings encriptados

5.4 **Mantenibilidad**

Métricas de Código

Complejidad Ciclomática: < 10 por método

Acoplamiento: Bajo (uso de interfaces)

Cohesión: Alta (responsabilidades claras)

Duplicación: < 3% (DRY principle)

Facilidades de Mantenimiento

Logging Estructurado: Serilog con contexto enriquecido

Configuración Centralizada: Un solo archivo de configuración

Documentación: Comentarios XML en todas las interfaces públicas

Nombres Descriptivos: Self-documenting code

Tecnologías y Justificación

6.1 **Stack Tecnológico**

Tecnología	**Versión**	**Justificación**
.NET	8.0	LTS, alto rendimiento, multiplataforma
Entity Framework Core	8.0	ORM maduro, migrations, múltiples proveedores
Serilog	Latest	Logging estructurado, múltiples sinks
CsvHelper	Latest	Parsing eficiente, streaming support
Polly	Latest	Resilience patterns, circuit breaker
MySQL	8.0	Requisito del proyecto, alto rendimiento
SQLite	Latest	Desarrollo local sin dependencias
Docker	Latest	Despliegue consistente, escalabilidad

6.2 Decisiones Clave

¿Por qué Worker Service?

Procesamiento en background

Gestión de ciclo de vida integrada

Soporte nativo para DI

Ideal para procesos ETL recurrentes

¿Por qué Entity Framework Core?

Abstracción de base de datos

Migraciones automáticas

LINQ support

Cambio fácil entre proveedores

¿Por qué Serilog?

Structured logging

Performance superior

Múltiples destinos (archivo, consola, cloud)

Contexto enriquecido
