داکیومنت تحلیل و رعایت اصول SOLID در پروژه DocumentVersioningAPI

مقدمه

پروژه **DocumentVersioningAPI** یک سرویس تحت وب برای مدیریت اسناد و نسخه‌های آن‌هاست که از تکنولوژی‌های زیر استفاده می‌کند:

- ASP.NET Core Web API
- Entity Framework Core
- Repository Pattern
- Dependency Injection (DI)
- Async/Await
- **SOLID**اصول مهندسی نرم‌افزار

هدف این مستند، بررسی دقیق نحوه پیاده‌سازی اصول **SOLID** در این پروژه است.

Single Responsibility Principle (SRP)

هر کلاس باید فقط یک دلیل برای تغییر داشته باشد.

پیاده‌سازی در پروژه:

- DocumentRepository: مسئولیتش فقط دسترسی به داده‌های Document است و هیچ منطق تجاری در آن دیده نمی‌شود.
- DocumentService: منطق مدیریت اسناد (ایجاد، اضافه‌کردن نسخه، دریافت لیست) را انجام می‌دهد و درگیر جزییات ذخیره‌سازی دیتابیس یا فایل نیست.
- FileSystemStorageService *(در نسخه Async)*: فقط مسئول ذخیره فیزیکی فایل‌هاست.

```
public class DocumentService
{
    private readonly IDocumentRepository _documentRepository;
    private readonly IStorageService _storageService;

    public DocumentService(IDocumentRepository documentRepository, IStorageService storageService)
    {
        _documentRepository = documentRepository;
        _storageService = storageService;
    }

    public async Task<Document> CreateDocumentAsync(string title, string description)
    {
        var document = new Document { Title = title, Description = description };
        await _documentRepository.AddAsync(document, default);
        return document;
    }
}
```

DocumentService :منطق ثبت سند را دارد؛ ذخیره فایل و دسترسی دیتابیس جدا شده‌اند.

Open/Closed Principle (OCP)

کلاس‌ها باید برای توسعه باز و برای تغییر بسته باشند.

پیاده‌سازی در پروژه:

- استفاده از اینترفیس‌ها مثل IRepository<T> و IStorageService باعث شده تغییر پیاده‌سازی راحت باشد بدون تغییر کد مصرف‌کننده.
- اگر بخواهیم ذخیره‌سازی ابری (مثل AWS S۳) اضافه کنیم، یک کلاس S۳StorageService می‌نویسیم که IStorageService را پیاده کند؛ کد سرویس‌ها تغییر نمی‌کند.

```
public interface IStorageService
{
    Task<string> SaveFileAsync(Stream fileStream, string fileName);
}
```

هر پیاده‌سازی جدید (لوکال، ابری، دیتابیس) این قرارداد را رعایت می‌کند و نیازی به تغییر DocumentService نیست.

Liskov Substitution Principle (LSP)

هر زیرکلاس باید بتواند جایگزین کلاس پایه شود بدون تغییر رفتار درست سیستم.

پیاده‌سازی در پروژه:

- DocumentRepository از Repository<Document> ارث‌بری کرده و می‌تواند در هر جایی که IRepository<Document> نیاز داریم استفاده شود بدون شکستن وابستگی‌ها.
- IStorageService هر پیاده‌سازی‌ای داشته باشد (FileSystemStorageService یا Mock برای تست) بدون مشکل کار می‌کند.

```
public class DocumentRepository : Repository<Document>, IDocumentRepository
{
    public Document GetByIdWithVersions(Guid id)
    {
        return Table.Include(d => d.Versions).FirstOrDefault(d => d.Id == id);
    }
}
```

می‌توان به‌جای DocumentRepository از FakeDocumentRepository در تست‌ها استفاده کرد بدون تغییر سایر بخش‌ها.

Interface Segregation Principle (ISP)

هیچ کلاسی نباید مجبور به پیاده‌سازی متدهایی شود که از آن‌ها استفاده نمی‌کند.

پیاده‌سازی در پروژه:

- IStorageService فقط یک متد SaveFileAsync دارد. این یعنی کلاس ذخیره‌سازی فقط مسئول همین تک کار است و مجبور به پیاده‌سازی چیزهای اضافه نیست.
- قرارداد CRUD کلی در IRepository<T> پیاده‌سازی شده که فقط متدهای مشترک موجودیت‌ها را دارد.

```
public interface IDocumentRepository : IRepository<Document>
{
    Document GetByIdWithVersions(Guid id);
}
```

اینترفیس اختصاصی فقط متد اضافی مربوط به Document را دارد و سایر موجودیت‌ها مجبور به پیاده‌سازی آن نیستند.

Dependency Inversion Principle (DIP)

کلاس‌های سطح بالا نباید به پیاده‌سازی‌های سطح پایین وابسته باشند، بلکه باید به اینترفیس‌ها وابسته باشند.

پیاده‌سازی در پروژه:

```
[1] DocumentsController
    ↓ وابسته به
[2] DocumentService
    ↓ وابسته به
        ├─ IDocumentRepository  →  (پیاده‌سازی: DocumentRepository)
        └─ IStorageService      →  (پیاده‌سازی: FileSystemStorageService)
[3] Program.cs / DI Container → ثبت و تزریق وابستگی‌ها
```

نمونه رجیستر کردن در DI:

```
builder.Services.AddScoped<IDocumentRepository, DocumentRepository>();
builder.Services.AddScoped<IStorageService, FileSystemStorageService>();
```

کد سرویس و کنترلر از پیاده‌سازی بی‌خبر است و فقط قرارداد (Interface) را می‌شناسد.

نتیجه‌گیری

در این پروژه اصول SOLID به شکل زیر رعایت شده است:

- SRP → جداسازی مسئولیت‌ها بین Repository، Service و Storage
- OCP → امکان گسترش و اضافه‌کردن پیاده‌سازی‌های جدید بدون تغییر کد موجود
- LSP → استفاده ایمن از کلاس‌های مشتق‌شده به جای کلاس پایه
- ISP → قراردادهای کوچک و هدفمند، جلوگیری از متدهای اضافه
- DIP → وابستگی به اینترفیس‌ها به جای کلاس‌های پیاده‌ساز

به لطف این اصول:

- کد خواناتر شده
- امکان تست‌پذیری بالا فراهم شده (Mockکردن سرویس‌ها)
- تغییرات آینده با هزینه کم قابل انجام است.