Aşağıda, “Controller → Service → Repository” üç katmanlı mimarinin her bir adımını ve verdiğim Dapper örneğini adım adım açıklıyorum. Amacım, bir bilgisayar mühendisliği öğrencisinin hem *neden* bu katmanları kullanırız, hem de *bu kod tam olarak ne yapıyor* sorularına yanıt vermesi.

---

1. Neden Katmanlı Mimari?

1. **Tekrar Kullanılabilirlik & Ayrık Sorumluluk**
    - Controller: HTTP isteğini alır, doğrulama ve dosya işleme yapar.
    - Service: İş (business) kurallarını (ör. ek doğrulamalar, başka API’lere çağrı) kapsüller.
    - Repository: Veritabanı okuma/yazma işlemlerini tek bir yerde toplar.
2. **Bakım Kolaylığı**
    - Veritabanı erişim kodunu değiştirmek isterseniz (ör. EF Core’dan Dapper’a geçiş), sadece repository’e dokunursunuz. Controller ve Service aynı kalır.
3. **Test Edilebilirlik**
    - Her katmanı bağımsız unit-test’lere tabi tutabilirsiniz. Örneğin, Service içine “fake” repository enjekte edip iş mantığını test edersiniz.

---

2. DapperContext (IDbConnection Fabrikası)

```
public class DapperContext
{
    private readonly IConfiguration _configuration;
    private readonly string _connectionString;

    public DapperContext(IConfiguration configuration)
    {
        _configuration = configuration;
        _connectionString = _configuration.GetConnectionString("DefaultConnection");
    }

    public IDbConnection CreateConnection()
        => new SqlConnection(_connectionString);
}
```

- **Görev**: Veritabanı bağlantısı oluşturmak.
- **Neden**: Her repository doğrudan `new SqlConnection(...)` yerine bu sınıftan bağlantı alır. Böylece bağlantı dizgisini (`connectionString`) sadece bir yerde tanımlarız.
- **Çalışma Şekli**:
    1. `IConfiguration`’dan `DefaultConnection` adındaki connection string’i okur.
    2. `CreateConnection()` çağrıldığında yeni bir `SqlConnection` örneği döner.

---

3. Repository Katmanı

```
public interface IBilgiRepository
{
    Task<int> AddBilgiAsync(AdminBilgilendirmeEkleDto dto);
}

public class BilgiRepository : IBilgiRepository
{
    private readonly DapperContext _dbContext;
```

```
    public BilgiRepository(DapperContext dbContext)
    {
        _dbContext = dbContext;
    }

    public async Task<int> AddBilgiAsync(AdminBilgilendirmeEkleDto dto)
    {
        var sql = @"
    INSERT INTO dt_bilgiler
       (Baslik, Icerik, TarihBaslangic, TarihBitis, Kapak, Files)
    VALUES
       (@Baslik, @Icerik, @TarihBaslangic, @TarihBitis, @Kapak, @Files);
    SELECT CAST(SCOPE_IDENTITY() AS INT);
";
        using var connection = _dbContext.CreateConnection();
        var parameters = new
        {
            dto.Baslik,
            dto.Icerik,
            dto.TarihBaslangic,
            dto.TarihBitis,
            dto.Kapak,
            dto.Files
        };
        return await connection.QuerySingleAsync<int>(sql, parameters);
    }
}
```

- **Interface (IBilgiRepository)**:
  - Hangi işlemler var, ne imzaya sahipler? (`AddBilgiAsync` → yeni kayıt ekler, eklenen kaydın `ID`'sini döner.)
- **Somut Sınıf (BilgiRepository)**:
  1. DapperContext aracılığıyla `IDbConnection` alır.
  2. Bir SQL sorgusu tanımlar (INSERT + SCOPE_IDENTITY()).
  3. DTO'daki alanları anonim bir nesne (`parameters`) içine koyar.
  4. `QuerySingleAsync<int>` diyerek sorguyu çalıştırır ve eklenen satırın auto-increment ID'sini alır.

Not: `using var connection` ile `connection` işi biter bitmez otomatik kapanır.

---

4. Service Katmanı

```
public interface IBilgiService
{
    Task AddBilgiAsync(AdminBilgilendirmeEkleDto dto);
}

public class BilgiService : IBilgiService
{
    private readonly IBilgiRepository _repository;

    public BilgiService(IBilgiRepository repository)
    {
        _repository = repository;
    }

    public async Task AddBilgiAsync(AdminBilgilendirmeEkleDto dto)
    {
        // İleri seviye iş kuralları buraya gelebilir
        await _repository.AddBilgiAsync(dto);
    }
```

```
}
```

- **Görev**:
  - İş mantığını (örneğin, ek doğrulamalar, başka servis çağrıları, olay yayma) burada yaparsınız.
  - Bu örnekte, direkt repository çağırarak veri ekliyoruz.
- **Faydası**:
  - Controller'ın sadece HTTP/DTO dönüşümüne odaklanmasını sağlar.
  - Gelecekte iş kurallarınız değişse bile Controller etkilenmez; sadece Service'e müdahale yeter.

---

5. API Controller

```
[ApiController]
[Route("api/[controller]")]
public class BilgiController : ControllerBase
{
    private readonly IBilgiService _bilgiService;
    private readonly IConfiguration _configuration;

    public BilgiController(IBilgiService bilgiService, IConfiguration configuration)
    {
        _bilgiService = bilgiService;
        _configuration = configuration;
    }

    [HttpPost("sercan")]
    public async Task<IActionResult> AddBilgi(
        [FromHeader(Name = "KurumDisiApiKey")] string apiKey,
        [FromForm] AdminBilgilendirmeEkleDto bilgiAddDto)
    {
        // 1) API key doğrulaması
        var validApiKey = _configuration["ApiKey"];
        if (apiKey != validApiKey)
            return Unauthorized("E005 Hata Oluştu");

        // 2) ModelState (DTO validasyonu)
        if (!ModelState.IsValid)
            return BadRequest(ModelState);

        // 3) Resim yükleme → DTO.Kapak alanını set et
        if (bilgiAddDto.Resim != null)
        {
            using var ms = new MemoryStream();
            await bilgiAddDto.Resim.CopyToAsync(ms);
            var ext = Path.GetExtension(bilgiAddDto.Resim.FileName);
            var uniqueName = $"{Guid.NewGuid()}{ext}";
            var path = Path.Combine("wwwroot/images", uniqueName);
            System.IO.File.WriteAllBytes(path, ms.ToArray());
            bilgiAddDto.Kapak = uniqueName;
        }

        // 4) Birden çok dosya yükleme → DTO.Files'a JSON string
        if (bilgiAddDto.Dosyalar != null && bilgiAddDto.Dosyalar.Any())
        {
            var allowed = new[] { ".pdf", ".docx", ".xls" };
            var list = new List<object>();
            foreach (var f in bilgiAddDto.Dosyalar)
            {
                var ext = Path.GetExtension(f.FileName).ToLower();
                if (!allowed.Contains(ext))
                    return BadRequest($"Yalnızca pdf, docx ve xls: {f.FileName}");

                using var ms = new MemoryStream();
```

```
                await f.CopyToAsync(ms);
                var uniqueName = $"{Guid.NewGuid()}{ext}";
                var docPath = Path.Combine("wwwroot/documents", uniqueName);
                System.IO.File.WriteAllBytes(docPath, ms.ToArray());

                list.Add(new
                {
                    OriginalFileName = f.FileName,
                    UniqueFileName = uniqueName
                });
            }
            bilgiAddDto.Files = JsonConvert.SerializeObject(list);
        }
        else
        {
            bilgiAddDto.Files = null;
        }

        // 5) Service'e devret
        try
        {
            await _bilgiService.AddBilgiAsync(bilgiAddDto);
            return Ok();
        }
        catch (Exception ex)
        {
            // Service veya Repository'deki beklenmedik hatayı yakala
            return StatusCode(500, $"Hata: {ex.Message}");
        }
    }
}
```

Akış Özeti

1. **İstek alınıyor** (`[HttpPost("sercan")]`).
2. **API Key** ve **DTO validasyonu** yapılıyor.
3. **Dosyalar fiziksel diske** (`wwwroot/images`, `wwwroot/documents`) kaydediliyor.
4. **DTO**'nun içinde dosya adları (`Kapak`, `Files`) set ediliyor.
5. **Service** katmanına DTO gönderiliyor.
6. Service de **Repository**'i çağırıyor ve Dapper ile veritabanına INSERT yapılıyor.
7. Sonuç "200 OK" veya hata kodu dönüyor.

---

6. Özet ve İpuçları

- **DTO vs Entity**: Burada DTO (`AdminBilgilendirmeEkleDto`) doğrudan repository'e gidiyor. Daha ileri senaryolarda, DTO'yu bir domain/entity objesine çevirip öyle göndermek isteyebilirsiniz.
- **Transaction**: Birden fazla tabloya ekleme yapacaksanız, Dapper üzerinden transaction kullanabilirsiniz:
- `using var tran = connection.BeginTransaction();`
- `await connection.ExecuteAsync(sql1, p1, tran);`
- `await connection.ExecuteAsync(sql2, p2, tran);`
- `tran.Commit();`
- **Hata Yönetimi**: Controller'da `try/catch` ile yakaladığınız hatayı log'layıp, kullanıcıya uygun mesaj dönebilirsiniz.
- **Test**:
    - Controller'u integration test ile, gerçek HTTP çağrısı yaparak test edin.
    - Service ve Repository'yi unit-test ile mock'lu bağımlılıklar üzerinden test edin.

Bu adımlarla “Controller → Service → Repository” katmanlı yapısını hem **neden** kullanacağınızı hem de **nasıl** uygulayacağınızı görmüş oldunuz. Sorularınız varsa detaylandırabiliriz!

---

ayrıntılı dosya yükleme açıklaması:
Controller’daki dosya yükleme (hem tek resim hem de birden fazla doküman) bölümü şu adımlardan oluşuyor. Her adımı tek tek ele alalım:

---

1. DTO’da IFormFile Özelliği

```
public class AdminBilgilendirmeEkleDto
{
    public IFormFile? Resim { get; set; }
    public List<IFormFile>? Dosyalar { get; set; }
    // Diğer alanlar: Baslik, Icerik, Tarih…, vs.
}
```

- **IFormFile** ASP.NET Core’da gelen HTTP isteğinde form-data ile yollanan dosyayı temsil eder.
- Tek bir dosya için `IFormFile`, birden fazla dosya için `List<IFormFile>` kullanırız.

---

2. Controller Metodunun İmzâsı

```
[HttpPost("sercan")]
public async Task<IActionResult> AddBilgi(
    [FromHeader(Name = "KurumDisiApiKey")] string apiKey,
    [FromForm] AdminBilgilendirmeEkleDto bilgiAddDto)
{ ... }
```

- `[FromForm]` der ki: “HTTP body’de `multipart/form-data` formatında gelen alanları DTO’ya bağla.”
- Böylece `bilgiAddDto.Resim` ve `bilgiAddDto.Dosyalar` otomatik olarak dolmuş olur.

---

3. Tek Resim (Kapak) Yükleme

```
if (bilgiAddDto.Resim != null)
{
    using var ms = new MemoryStream();
    await bilgiAddDto.Resim.CopyToAsync(ms);

    // 1. Dosya uzantısını al
    var ext = Path.GetExtension(bilgiAddDto.Resim.FileName);

    // 2. Çakışma olmasın diye benzersiz bir isim üret
    var uniqueName = $"{Guid.NewGuid()}{ext}";

    // 3. Fiziksel kaydedilecek dizini oluştur
    var path = Path.Combine("wwwroot/images", uniqueName);

    // 4. Baytları diske yaz
    System.IO.File.WriteAllBytes(path, ms.ToArray());

    // 5. DTO’ya kaydedilen ismi ata ki DB’ye yazabilesin
    bilgiAddDto.Kapak = uniqueName;
```

```
}
```

Adım Adım Açıklama

1. **Null kontrolü**
   - `bilgiAddDto.Resim != null` ⇒ kullanıcı gerçekten bir dosya yollamış mı?
2. **MemoryStream'e kopyalama**
   - `CopyToAsync(ms)` ile `IFormFile`'ın içindeki akışı (stream) `MemoryStream`'e aktarırız.
   - Bu sayede diske yazmak için dosyanın bütün baytlarını elimizde bir dizi olarak tutarız.
3. **Dosya uzantısını alma**
   - `Path.GetExtension(originalName)` ⇒ orijinal dosya adından ("foto.jpg" → ".jpg") uzantıyı çeker.
4. **GUID ile benzersiz isim oluşturma**
   - `Guid.NewGuid()` rastgele, düşük çakışma ihtimali olan bir ID üretir.
   - Bu sayede aynı adlı iki dosya yüklense bile isim çakışması olmaz.
5. **Hedef yolun birleştirilmesi**
   - `Path.Combine("wwwroot/images", uniqueName)`
   - `wwwroot` klasörü, ASP.NET Core'da statik dosyaların (resim, css, js) servis edildiği kök klasördür.
6. **Dosyayı diske yazma**
   - `File.WriteAllBytes(path, byteArray)` → tüm baytları tek seferde fiziksel dosyaya yazar.
7. **DTO'ya kaydedilen adı atama**
   - Böylece `bilgiAddDto.Kapak`'ta fiziksel isim kalıcı hale gelir ve Dapper ile DB'ye INSERT ederken bu alanı kullanabilirsiniz.

---

4. Birden Fazla Doküman (Dosyalar) Yükleme

```
if (bilgiAddDto.Dosyalar != null && bilgiAddDto.Dosyalar.Any())
{
    var allowed = new[] { ".pdf", ".docx", ".xls" };
    var list = new List<object>();

    foreach (var f in bilgiAddDto.Dosyalar)
    {
        var ext = Path.GetExtension(f.FileName).ToLower();
        if (!allowed.Contains(ext))
            return BadRequest($"Yalnızca pdf, docx ve xls: {f.FileName}");

        using var ms = new MemoryStream();
        await f.CopyToAsync(ms);

        var uniqueName = $"{Guid.NewGuid()}{ext}";
        var docPath = Path.Combine("wwwroot/documents", uniqueName);
        System.IO.File.WriteAllBytes(docPath, ms.ToArray());

        list.Add(new
        {
            OriginalFileName = f.FileName,
            UniqueFileName = uniqueName
        });
    }

    bilgiAddDto.Files = JsonConvert.SerializeObject(list);
}
else
{
    bilgiAddDto.Files = null;
}
```

Detaylı Açıklama

1. **Null ve boş liste kontrolü**
   - Eğer hiç dosya gelmediyse (`null` veya `Any()==false`), `Files` alanını `null` yaparız.
2. **İzin verilen uzantılar**
   - Burada sadece `.pdf`, `.docx`, `.xls` kabul ediliyor.
   - Her dosya için `Path.GetExtension` ile uzantıyı küçük harfe çevirip kontrol ederiz.
3. **Her dosya için:**
   - **Stream kopyalama** (`CopyToAsync(ms)`) ve **benzersiz isim** üretme (`Guid.NewGuid()` + uzantı).
   - **Fiziksel kaydetme** → `wwwroot/documents` altına.
   - **Orijinal ve sunucu ismini** bir nesne olarak `list`'e ekleme.
4. **JSON'a çevirme**
   - `JsonConvert.SerializeObject(list)` →
   - `[`
   - `  { "OriginalFileName":"dosya1.pdf", "UniqueFileName":"a3f1-... .pdf" },`
   - `  { "OriginalFileName":"rapor.xls",  "UniqueFileName":"b9d2-... .xls" }`
   - `]`
   - Bu JSON string'i `Files` alanına atıyoruz. Veritabanında `nvarchar(max)` gibi bir alanda saklanabilir.

---

5. Neden Böyle Yapıyoruz?

- **Tek bir alanda JSON** tutmak:
  - ayrı bir "Dosyalar" tablosu yerine, dosya bilgilerini tek satırda saklarsınız.
- **Statik dosya servisi**:
  - `wwwroot` altına koyduğunuzda, web sunucu (IIS/Kestrel) otomatik olarak `/images/unique.jpg` veya `/documents/unique.pdf` URL'leriyle erişime açar.
- **GUID kullanımı**:
  - Dosya adlarının çakışmaması ve güvenlik için rastgele isim tercih edilir.

---

6. Özet

1. `[FromForm]` ile gelen `IFormFile`'ı yakalayın.
2. **MemoryStream**'e kopyalayarak dosya içeriğine erişin.
3. **Benzersiz isim** oluşturun (GUID + uzantı).
4. **Path.Combine** ile wwwroot altındaki klasöre tam yol oluşturun.
5. **File.WriteAllBytes** ile diske kaydedin.
6. Resim için DTO'ya `Kapak`, birden fazla dosya için JSON string'i `Files` alanına set edin.
7. Sonrasında DTO'nuzu servis/repository katmanına gönderin ve DB'ye yazdırın.

Bu adımları izleyerek ASP.NET Core'da güvenli, çakışmasız ve izlenebilir dosya yükleme işlemi gerçekleştirmiş olursunuz.