# Katmanlı Mimari - Proje Rehberi (Senaryo Anlatımı)

## Entities Katmanı (Varlıklar)

✔ Amaç:

Veritabanına denk gelen sınıfları tanımlar. Her tablo bir class'tır.

✔ Örnek:

```csharp

public class Application

{

public int Id { get; set; }

public int ApplicantId { get; set; }

public int BootcampId { get; set; }

public ApplicationState ApplicationState { get; set; }

}

public enum ApplicationState

{

PENDING,

APPROVED,

REJECTED,

IN\_REVIEW,

CANCELLED

}

```

## Repository Katmanı (Veri Erişimi)

✔ Amaç:

Veritabanı işlemlerini soyutlamak. EF Core'a bağlılı olmayan arayüzler yazmak.

✔ Dosyalar:

IAsyncRepository.cs:

```csharp

public interface IAsyncRepository<T> where T : class

{

Task<T> AddAsync(T entity);

Task<T> DeleteAsync(T entity);

Task<T> UpdateAsync(T entity);

Task<T> GetByIdAsync(int id);

Task<IReadOnlyList<T>> GetAllAsync();

}

```

EfRepositoryBase.cs:

```csharp

public class EfRepositoryBase<T> : IAsyncRepository<T> where T : class

{

protected readonly DbContext \_context;

public EfRepositoryBase(DbContext context)

{

\_context = context;

}

public async Task<T> AddAsync(T entity)

{

await \_context.Set<T>().AddAsync(entity);

await \_context.SaveChangesAsync();

return entity;

}

// Diğer CRUD metotları da benzer

}

```

BootcampDbContext.cs:

```csharp

public class BootcampDbContext : DbContext

{

public DbSet<Application> Applications { get; set; }

public DbSet<Bootcamp> Bootcamps { get; set; }

public DbSet<Blacklist> Blacklists { get; set; }

public BootcampDbContext(DbContextOptions<BootcampDbContext> options) : base(options) {}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.ApplyConfigurationsFromAssembly(typeof(BootcampDbContext).Assembly);

base.OnModelCreating(modelBuilder);

}

}

```

ApplicationConfiguration.cs:

```csharp

public class ApplicationConfiguration : IEntityTypeConfiguration<Application>

{

public void Configure(EntityTypeBuilder<Application> builder)

{

builder.HasKey(x => x.Id);

builder.Property(x => x.ApplicationState).HasConversion<string>();

builder.Property(x => x.ApplicantId).IsRequired();

}

}

```

## Business Katmanı (Servis / İş Mantığı)

✔ Amaç:

Veri işleme, karar verme, doğrulama gibi uygulama mantığını yazarız.

✔ Yapı:

- IApplicationService.cs: Ne iş yapacak?

- ApplicationManager.cs: Nasıl yapacak?

- ApplicationCreateRequest.cs: Kullanıcının göndereceği veri

- ApplicationResponse.cs: Kullanıcıya geri dönecek veri

## WebAPI Katmanı (Controller / Endpoint)

✔ Amaç:

Kullanıcının sistemle iletişimini sağlamak. REST API uçları (endpoint) tanımlar.

✔ Örnek:

```csharp

[ApiController]

[Route("api/[controller]")]

public class ApplicationController : ControllerBase

{

private readonly IApplicationService \_applicationService;

public ApplicationController(IApplicationService applicationService)

{

\_applicationService = applicationService;

}

[HttpPost]

public async Task<IActionResult> Add(ApplicationCreateRequest request)

{

var response = await \_applicationService.AddAsync(request);

return Ok(response);

}

}

```

## Hatırlamanı kolaylaştıracak ipucu

- Entities: Tablo gibi düşün.

- Repository: Veritabanına git-gel yapan kapı.

- Business: Beyin, karar veren yer.

- WebAPI: Kullanıcı ile sistem arasındaki köprü.

Her katmanın tek sorumluluğu var, bu seni projede kaybolmaktan kurtarır.