

Here is an enhanced version of the Clean Architecture cheat sheet for ASP.NET Core, now including examples for each component and layer dependencies.

---

# Clean Architecture Layers Cheat Sheet with Examples and Dependencies

## 1 Domain Layer

**Purpose:** Core business logic, business rules, entities.

**Typical contents & examples:** - **Entities:** Represent business objects with invariants.

```
public class Country
{
    public int Id { get; private set; }
    public string Name { get; private set; }
    public string FlagUri { get; private set; }

    public Country(string name, string flagUri)
    {
        if(string.IsNullOrWhiteSpace(name)) throw new DomainException("Name required");
        if(!Uri.IsWellFormedUriString(flagUri, UriKind.Absolute)) throw new DomainException("Invalid FlagUri");
        Name = name;
        FlagUri = flagUri;
    }
}
```

- **Value Objects:** Immutable types.

```
public record Email(string Value)
{
    public Email
    {
        if(!Regex.IsMatch(Value, @"^[\w\.-]+@[^\w\.-]+\.\w{2,}$")) throw new DomainException("Invalid email");
    }
}
```

- **Domain Services:** Business logic not tied to a single entity.

```
public class CountryValidatorService
{
    public bool IsNameUnique(string name, IEnumerable<Country> countries) => !
countries.Any(c => c.Name == name);
}
```

- **Domain Exceptions:** Custom exceptions.

```
public class DomainException : Exception { public DomainException(string msg) :
base(msg) {} }
```

- **Interfaces:** Contracts for repositories.

```
public interface ICountryRepository { Task<int> AddAsync(Country country); }
```

**Dependencies:** None (independent core layer).

---

## 2 Application Layer (Service Layer)

**Purpose:** Orchestrates use cases, handles DTOs, validation.

**Typical contents & examples:** - DTOs:

```
public class CreateCountryDto { public string Name { get; set; } = null!;
public string FlagUri { get; set; } = null!; }
```

- **Validators (FluentValidation):**

```
public class CreateCountryDtoValidator : AbstractValidator<CreateCountryDto>
{
    public CreateCountryDtoValidator()
    {
        RuleFor(x => x.Name).NotEmpty().MaximumLength(100);
        RuleFor(x => x.FlagUri).Must(uri => Uri.IsWellFormedUriString(uri,
UriKind.Absolute));
    }
}
```

- **Application Services:**

```

public class CountryService
{
    private readonly ICountryRepository _repo;
    public CountryService(ICountryRepository repo) => _repo = repo;

    public async Task<int> CreateAsync(CreateCountryDto dto)
    {
        var country = new Country(dto.Name, dto.FlagUri);
        return await _repo.AddAsync(country);
    }
}

```

- **Mappers:**

```

public static class CountryMapper
{
    public static CountryDto ToDto(this Country c) => new CountryDto { Id =
c.Id, Name = c.Name, FlagUri = c.FlagUri };
}

```

- **Application Exceptions:**

```

public class NotFoundException : Exception { public NotFoundException(string
msg) : base(msg) {} }

```

**Dependencies:** Depends on Domain Layer.

---

## 3 Infrastructure Layer

**Purpose:** Persistence, external services, framework implementations.

**Typical contents & examples:** - EF Core DbContext & Repositories:

```

public class AppDbContext : DbContext { public DbSet<Country> Countries { get;
set; } = null!; }

public class CountryRepository : ICountryRepository
{
    private readonly AppDbContext _db;
    public CountryRepository(AppDbContext db) => _db = db;
    public async Task<int> AddAsync(Country country)
    {

```

```

        await _db.Countries.AddAsync(country);
        await _db.SaveChangesAsync();
        return country.Id;
    }
}

```

- **External services:** HTTP clients, Redis, file storage. **Dependencies:** Depends on Domain Layer (interfaces) and optionally Application Layer.
- 

## 4 API / Presentation Layer

**Purpose:** Expose endpoints, handle requests/responses.

**Typical contents & examples:** - **Controllers / Minimal APIs:**

```

app.MapPost("/countries", async (CreateCountryDto dto,
IValidator<CreateCountryDto> validator, CountryService service) =>
{
    var result = await validator.ValidateAsync(dto);
    if(!result.IsValid) return Results.ValidationProblem(result.ToDictionary());
    return Results.Created("/countries", await service.CreateAsync(dto));
});

```

- **Request/Response DTOs** (already shown) - **Middleware:** Logging, exception handling, JWT authentication. **Dependencies:** Depends on Application Layer (services, DTOs, validators).
- 

## 5 Cross-Cutting Concerns

**Purpose:** Shared features across layers.

**Typical contents & examples:** - **Logging:** Serilog, ILogger - **Caching:** Redis, MemoryCache - **Authentication & Authorization:** JWT, roles - **Exception handling middleware:**

```
app.UseMiddleware<ExceptionHandlingMiddleware>();
```

- **Mapping utilities:** AutoMapper profiles or manual mapping **Dependencies:** Can depend on Domain & Application layers.
-

## 6 Testing Layer

**Purpose:** Unit & integration tests

**Typical contents & examples:** - Unit tests for entities, domain services - Integration tests for EF Core, repositories - Mocking external services - Test data builders

**Dependencies:** Depends on all layers it tests.

---

## ✓ Layer Dependencies Summary

Layer	Depends On
Domain	None
Application	Domain
Infrastructure	Domain, Application (optional)
API / Presentation	Application, Domain (via Application)
Cross-Cutting	Domain, Application (optional)
Testing	Domain, Application, Infrastructure, API

---

This updated cheat sheet now includes **examples for each component** and **layer dependencies**, making it a practical guide for building a complete, production-ready Clean Architecture ASP.NET Core Web API.