# Tiered Structure

Browser/Client 🡪 Web Server (Web App) 🡪 API Server (HTTP API) 🡪 Database Server.

# API n-Layered Architecture

Layers, Components & project dependencies

Include Unit test project and DB Project in the solution.

[Chapter 10: Component Guidelines](https://msdn.microsoft.com/en-us/library/ee658121.aspx#Overview)

[Business Logic](https://inviqa.com/blog/architecture-patterns-domain-model-and-friends) (Organising Domain Logic) - [Domain Logic and SQL](https://martinfowler.com/articles/dblogic.html#ComplexQueries)

Domain Logic Patterns

* **Transaction script pattern** – all the logic in single function/Method (or procedure) for particular transaction, directly call Database/Thin DB Wrapper. User when business logic is not too complicated.
* Table module pattern -
* Domain model pattern (DDD) - Multiple objects and each of them handles a slice of domain logic. Is an object model of domain that incorporates bot behaviour and data?
  + Simple Domain model – Domain object = DB table
    - Uses Active Record pattern
  + Rich Domain Model
    - Requires Data Mapper pattern
* Service Layer pattern
  + Is placed over underlying Domain model or Table module
* Logic in SQL - Logic in DB Stored procedures
  + Row Data Gateway
  + Table Data Gateway

## Presentation/Services

Async and Await Endpoints

Long running tasks – 202 Accepted e.g. file upload

Global Exception Handling and logging with Serilog

Swagger and Health Checks - [AspNetCore.Diagnostics.HealthChecks](https://github.com/xabaril/AspNetCore.Diagnostics.HealthChecks)

**Components**

* **<<APITProjectname>>.API** (REST & HTTP), <<APITProjectname>>.gRPC, or <<APITProjectname>>. **GraphQL**
* **<<APITProjectname>>.APIModels** – ***ViewModel***
  + Request and Response DTOs (View Models)
    - [FluentValidation](https://docs.fluentvalidation.net/en/latest/) for request validation. And returns BadRequest if fails
    - Automapper to convert DataModels to Response - [Web API Design Anti-Pattern: Exposing your database model](https://shekhargulati.com/2021/10/15/web-api-design-anti-pattern-exposing-your-database-model/)
      * [Your data model is not an API](https://tyk.io/blog/your-data-model-is-not-an-api/)
  + Response and Error abstraction layer

## Business Layer

[**Domain logic pattern**](https://inviqa.com/blog/architecture-patterns-domain-model-and-friends) - The pattern provides an object-oriented way of dealing with complicated logic. Instead of having one procedure that handles all business logic for a user action there are multiple objects and each of them handles a slice of domain logic that is relevant to it

**Components**

* **<<APITProjectname>>.Business** or<<APITProjectname>>.Manager
* <<APITProjectname>>.BusinessRules

[Which side is more logical: front-end or back-end?](https://www.quora.com/Which-side-is-more-logical-front-end-or-back-end)

1. Business logic  
2. View Logic

* Business Logic: It deals with the security of your application (Authentication and Authorisation) and the actual data. This is the backend of your app. Business logic should never appear in your front end because your front ends can be multiple (viz. Android, HTML, iOS), and if you write your business logic on front end, you will have to replicate it on all front ends. This is a severe code smell.
* View Logic: It deals with GUI elements. For a good, elegant and intuitive GUI, you need to write a good view logic at your front end. This is important as well because customers directly see and interact with the front end of your app.

## Data/Persistence

[Data access object](https://en.wikipedia.org/wiki/Data_access_object)

[Database abstraction layer](https://en.wikipedia.org/wiki/Database_abstraction_layer)

[Object–relational mapping](https://en.wikipedia.org/wiki/Object%E2%80%93relational_mapping)

**Transaction Script pattern**- Transaction Script organizes all this logic primarily as a single procedure, making calls directly to the database or through a thin database wrapper. It is not object-oriented at all

**Components**

* **<<APITemplate>>.DataAccess**
  + EF with Dapper Repository - DBContext
  + [Repository Pattern](https://code-maze.com/net-core-web-development-part4/) – (Optional -Generic Repository with Unit of work)
* **<<APITProjectname>>.DataModel**
  + **\***[**strongly-typed (entity) IDs**](https://andrewlock.net/using-strongly-typed-entity-ids-to-avoid-primitive-obsession-part-1/)
    - **Directly using guid in entity leads to bugs**

|  |
| --- |
| ~~public class Order~~  ~~{~~  ~~public Guid Id { get; set; }~~  ~~}~~  ~~public class OrderItem~~  ~~{~~  ~~public Guid Id { get; set; }~~  ~~}~~  ~~public void Processorder(Guid orderId) { }~~  ~~Processorder(order.Id);~~  ~~Processorder(orderItem.Id);~~ |
| * **Use Strongly Typed IDs and Compiler Will Save You From Bugs**   public readonly record struct OrderId(Guid Value);  public readonly record struct OrderItemId(Guid Value);  public class Order  {  public OrderId Id { get; set; }  }  public class OrderItem  {  public OrderItemId Id { get; set; }  }  var orderId = new OrderId(Guid.NewGuid());  var orderItemId = new OrderItemId(Guid.NewGuid());  Processorder(orderItemId); |

* <<APITProjectname>>.DbMigrator
* Infrastructure – Even DataAccess can be included like ORM/Data Integrations
  + <https://docs.abp.io/en/abp/latest/Audit-Logging>

## Tests

* + <<APITProjectname>>.**API.Tests**
    - [Unit test](https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/testing?view=aspnetcore-6.0) and [Integration tests in ASP.NET Core](https://docs.microsoft.com/en-us/aspnet/core/test/integration-tests?view=aspnetcore-6.0)
    - Using xUnit, Moq etc.
  + <<APITProjectname>>.**DataAccess.Tests**
  + <<APITProjectname>>.**Business.Tests**

## Cross Cuttings or Common

**Components**

* <<APITProjectname>>.**Caching** -- Cache Abstraction like [EasyCaching](https://github.com/dotnetcore/EasyCaching)
  + Install Redis
* <<APITProjectname>>.**ExceptionHandling**
  + ?
* <<APITProjectname>>.**Localization**
  + ?
* <<APITProjectname>>.**Logging** -- logging Abstraction like [common-logging](https://github.com/net-commons/common-logging)
  + Install Serilog.AspNetCore
  + Install Serilog.Sinks.Console
  + Install [Serilog.Sinks.MSSqlServer](https://nuget.org/packages/serilog.sinks.mssqlserver)
  + Install Serilog.Sinks.Seq
* <<APITProjectname>>.**Mapper**
  + Install AutoMapper
  + Install AutoMapper.EF6
* <<APITProjectname>>.**Security**
* <<APITProjectname>>.**Validation**
  + Install FluentValidation.AspNetCore
    - Ark.Tools.Solid.FluentValidate
  + [ardalis](https://github.com/ardalis)/**[GuardClauses](https://github.com/ardalis/GuardClauses)**
  + Null checks, Validation Input and respond back Validation errors
    - Also present client-side validation errors
* <<APITProjectname>>.**Configuration**
  + [Secret manager](https://docs.microsoft.com/en-us/aspnet/core/security/app-secrets?view=aspnetcore-6.0&tabs=windows)
* <<APITProjectname>>.**Common**
  + [Constants class](https://softwareengineering.stackexchange.com/questions/230410/suggest-a-best-practice-to-create-constants-class)
    - [const](https://msdn.microsoft.com/en-us/library/e6w8fe1b.aspx) keyword: Compile time
    - static [readonly](https://www.c-sharpcorner.com/UploadFile/0c1bb2/read-only-and-constant-in-C-Sharp/): runtime -  settings from the config file
* Etc.

### Other aspects

* WebHooks
* Metric and monitoring
  + Health checks 🡪 Resource Health
    - EF or DB, URIs - External URL or file, and system.
  + APM
  + monitoring

## Dependency between layers or Code Flow

Presentation Layer 🡪 Application/Business Layer 🡪 Domain Layer 🡪 Infrastructure/Persistence/Database Layer

# DDD Architecture

DDD is set as a standard to develop different popular architectures, such as Onion Architecture, Clean Architecture, Hexagonal Architecture, etc.

* [ABP Framework DDD solution structure](https://docs.abp.io/en/abp/latest/Startup-Templates/Application#solution-structure)
* [Domain-Driven Design in ASP.NET Core applications](https://enlabsoftware.com/development/domain-driven-design-in-asp-net-core-applications.html)
* [Design a DDD-oriented microservice](https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/ddd-oriented-microservice)
* [Layered design in DDD](https://codilime.com/blog/what-is-domain-driven-design-and-how-can-it-benefit-your-product-development/#layered-design-in-ddd) 
  + [Implementing Domain Driven Design: Part I](https://tealfeed.com/implementing-domain-driven-design-part-mrxps)

[**A Template for Clean Domain-Driven Design Architecture**](https://blog.jacobsdata.com/2020/03/02/a-clean-domain-driven-design-architectural-template)

* Domain Model layer - Each domain entity
  + Application - Aggregate root corresponding to each use case
* Infrastructure Persistence layer - Each functional area of the operating system (file operations, etc.), manages storage and data access and/or external resources/services
  + [Design the infrastructure persistence layer](https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design)
  + Persistence - Each database table
* Presentation - Aggregate root corresponding to each screen or web page
* Cross-cutting (Common) - Each cross-cutting concern (Logging, Security, etc.)

## Layers In DDD

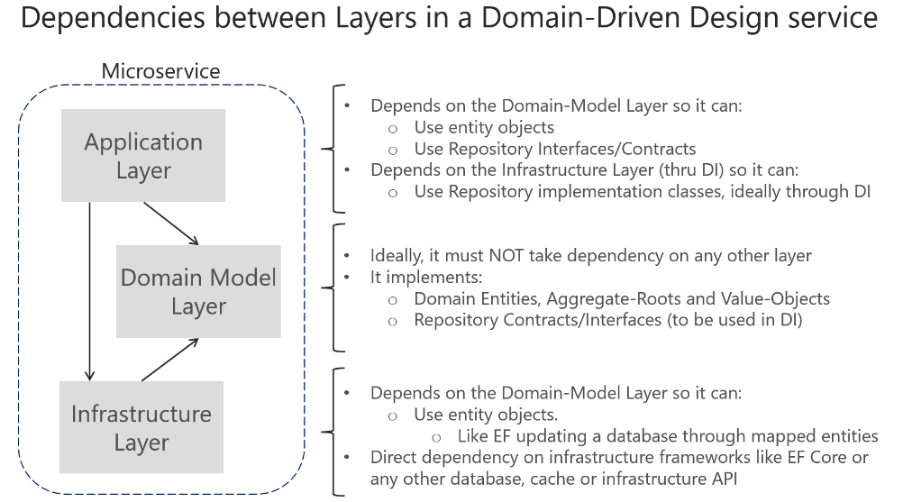
* **Domain Layer Building Blocks🡪 DDD Patterns**

A place to define logic concepts, principles, patterns, and behaviours of data, including domain validation, calculations, and expressions for system operations. Has Domain objects (entities and value objects)

[Persistence Ignorance](https://deviq.com/persistence-ignorance/) and the [Infrastructure Ignorance](https://ayende.com/blog/3137/infrastructure-ignorance) principles

* + **Entities**: POCO classes, construction, and model validation.
    - **Domain Entity Model (**anaemic domain model**)**
      * only hold data properties so not object-oriented design
      * POCO Entity Classes (Clean C# code Or Properties)
    - **Domain Entities with Data + Behaviour (**Rich domain model**)**
    - **POCO Entity Classes (Clean C# code Or properties)**
  + **Aggregates and Aggregate Root**: The rules, computation, logic of domains, and related objects when updating the domain. According to Martin Fowler, an aggregate is a cluster of domain objects that can be treated as a single unit.
  + **Value objects**: The value of an object related to Domain entities. In principle, Value Objects have no identity, and once been initialized, will not be modified. They can be understood as *immutable* classes.
  + **Interfaces**: They help define business behaviors, etc. Other layers will be responsible for implementing these definitions.
  + **Repository Interfaces/ServiceBase**: The Interfaces of generic repositories, domain repositories, and services. Other layers will inherit and develop them.
  + **ILogger/DTOs/Exceptions**: Notifications and information transferred to other services.
  + Others
* **Application Layer**
  + **Web API Services**
    - [EISK](https://github.com/EISK)/[eisk.webapi](https://github.com/EISK/eisk.webapi) (DDD and Clean architecture)
  + Client Apps
    - Mobile application
    - Web MVC/API application
    - Desktop application
    - IoT
  + Others services
* **Infrastructure persistence layer**
  + **Repositories**: Implement Repositories here, including Generic Repository and <Entity> Repository. And Unit of work
  + **Data access**: Contexts and the API connections link to databases.
    - SQL: ADO.NET, EntityFramework, Dapper, and ORM, etc.
    - In-Memory stores.
    - Caching, NoSQL, and so on.
    - Data seeding
* **User Interface Layer (Presentation Layer**)
* **Others/Common**
  + **Logging**
  + **Cryptography**

### Dependencies b/w DDD layers



### Other Concepts

* EventSourcing in DDD
  + EventSourcing library [Aggregates.NET](https://github.com/volak/Aggregates.NET) 🡪  [NServiceBus](https://github.com/Particular/NServiceBus) and [EventStore](https://github.com/EventStore/EventStore)
* CQRS
  + [jbogard](https://github.com/jbogard)/[MediatR](https://github.com/jbogard/MediatR)
* CQRS+ES
  + [eventflow](https://github.com/eventflow)/[EventFlow](https://github.com/eventflow/EventFlow)
  + [OpenCQRS](https://github.com/OpenCQRS/OpenCQRS) - .NET Core library for DDD, CQRS and Event Sourcing with Azure Service Bus integration
* MediatR + CQRS
* Complete DDD – Samples
  + [charlessolar](https://github.com/charlessolar)/**[eShopOnContainersDDD](https://github.com/charlessolar/eShopOnContainersDDD)**
  + [**clean-architecture-dotnet**](https://github.com/thangchung/clean-architecture-dotnet)
  + [DNC-DShop](https://github.com/devmentors)
  + [dotnet-architecture/**eShopOnContainers**](https://github.com/dotnet-architecture/eShopOnContainers) 🡪 [Implementation in .net core](https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/net-core-microservice-domain-model)
  + [rafaelfgx](https://github.com/rafaelfgx)/[**Architecture**](https://github.com/rafaelfgx/Architecture)

## Links and Books for DDD

* [An awesome guide on how to build RESTful APIs with ASP.NET Core](https://www.freecodecamp.org/news/an-awesome-guide-on-how-to-build-restful-apis-with-asp-net-core-87b818123e28/)
* <https://realtoughcandy.com/domain-driven-design-books/>
* [vladikk](https://github.com/vladikk)/[**awesome-ddd**](https://github.com/vladikk/awesome-ddd)
  + [vladikk](https://github.com/vladikk)/[learning-ddd](https://github.com/vladikk/learning-ddd)
* [heynickc](https://github.com/heynickc)/[**awesome-ddd**](https://github.com/heynickc/awesome-ddd)

# Clean Architecture

[Domain-Centric Architecture](https://medium.com/codex/clean-architecture-for-dummies-df6561d42c94)

Also Known as

* Onion Architecture
* Hexagonal Architecture (aka Ports and Adapters)

[EISK](https://github.com/EISK)/[eisk.webapi](https://github.com/EISK/eisk.webapi) (DDD and Clean architecture)

# Common

## API Technology

[Choosing An API Technology: GRPC, REST, GraphQL](https://speedscale.com/2021/07/20/choosing-an-api-technology-grpc-rest-graphql/)

[API Showdown: REST vs. GraphQL vs. gRPC – Which Should You Use?](https://www.infoq.com/podcasts/api-showdown-rest-graphql-grpc/)

## DB Access and ORM

Singleton connection. Use ORMs or ADO.NET

ORM

* EF core
  + Querying – LINQ query, lambda expression Or [Raw SQL Queries](https://docs.microsoft.com/en-us/ef/core/querying/raw-sql) (SPs too)
  + [Complex Query Operators](https://docs.microsoft.com/en-us/ef/core/querying/complex-query-operators) – Joins
  + [User-defined function mapping](https://docs.microsoft.com/en-us/ef/core/querying/user-defined-function-mapping)
* Dapper
  + High performance on reads.

CQRS

* Dapper - Read Side
* EF Core - Write Side

## Dependency Injection

For decoupled layers

Three Types of Dependency Injection

* Constructor Injection (also known as Type 3) \*\*\*\* Best way and used in C# .net 6
* [~~Property Injection~~](https://dotnettutorials.net/lesson/setter-dependency-injection-design-pattern-csharp/) ~~(aka setter injection) (also known as Type 2)~~
* ~~Interface Injection (also known as Type 1) - method injection~~

.NET6 C# Steps - [Dependency injection - .NET | Microsoft Learn](https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection)

1. Constructor Injection
2. Register the interfaces and classes in the container class
   1. [Service lifetimes](https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection#service-lifetimes)
      1. Transient - created each time they are requested from the service container and transient services are disposed at the end of the request. Best for lightweight, stateless services
      2. Scoped -  services are created once per client request (connection)
      3. Singleton - The first time they're requested
      4. [AddTransient, AddScoped and AddSingleton Services Differences](https://stackoverflow.com/questions/38138100/addtransient-addscoped-and-addsingleton-services-differences)

Option 1:

|  |
| --- |
| // Add a service to DI  services.AddSingleton<IGreeterService, GreeterService>(); |

Option 2: Automatically register services

|  |
| --- |
| services.Register(); //in ConfigureServices method  //And add the following class in the Startup.cs  public static class Injector  {  public static void Register(**this** IServiceCollection services)  {  services.AddTransient(IGreeterService, GreeterService);  }  } |
| (OR) we can have something like [**ServiceExtensions.cs**](https://github.com/fungainyatanga/ASP.NET-Core-Web-API/blob/master/AccountOwnerServer/Extensions/ServiceExtensions.cs)  [How to use ServiceExtensions in .Net Core?](https://stackoverflow.com/questions/53460593/how-to-use-serviceextensions-in-net-core)  [https://github.com/sunilkumarmedium/CleanArchitectureApp/blob/main/CleanArchitectureApp.WebApi/Startup.cs e.g](https://github.com/sunilkumarmedium/CleanArchitectureApp/blob/main/CleanArchitectureApp.WebApi/Startup.cs%20e.g). services.AddApplicationLayer(); |
| services.AddScopedSerivces (); //in ConfigureServices method  //Here class name can be anything and this keyword is mandatory  ~~public static class ServiceExtentions~~  ~~{~~  public static void AddScopedSerivces(**this** IServiceCollection services)  {  services.AddScoped(IGreeterService, GreeterService);  }  ~~}~~ |

[Stoyanov8](https://github.com/Stoyanov8)**/**[**Serviced**](https://github.com/Stoyanov8/Serviced)

Serviced is a simple lightweight library that handles service registrations for you

[khellang](https://github.com/khellang)/**[Scrutor](https://github.com/khellang/Scrutor)**

**Note:**

TryAddSingleton - which register the service only if there isn't already an implementation registered

**Transient**

* since they are created every time they will use **more memory** & Resources and can have a **negative** impact on performance
* use this for the **lightweight** service with little or **no state**.

**Scoped**

* better option when you want to maintain state within a request.

**Singleton**

* memory leaks in these services will build up over time.
* also memory efficient as they are created once reused everywhere.

Use Singletons where you need to maintain application wide state. Application configuration or parameters, Logging Service, caching of data is some of the examples where you can use singletons.

Injecting service with different lifetimes into another

**Never inject Scoped & Transient services into Singleton service.** ( This effectively converts the transient or scoped service into the singleton.)

**Never inject Transient services into scoped service** ( This converts the transient service into the scoped.)

**Common errors**

[Dependency Injection error: Unable to resolve service for type while attempting to activate, while class is registered](https://stackoverflow.com/questions/40900414/dependency-injection-error-unable-to-resolve-service-for-type-while-attempting)

**References:**

[Dependency injection - .NET | Microsoft Learn](https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection)

[The 3 Types of Dependency Injection (linkedin.com)](https://www.linkedin.com/pulse/3-types-dependency-injection-mohammad-ramezani)

[New dependency injection features in .NET 6 (andrewlock.net)](https://andrewlock.net/exploring-dotnet-6-part-10-new-dependency-injection-features-in-dotnet-6/)

## [Localization](https://stackoverflow.com/questions/45167350/localization-in-external-class-libraries-in-asp-net-core)

* [Localization in external class libraries in ASP.NET Core](https://stackoverflow.com/questions/45167350/localization-in-external-class-libraries-in-asp-net-core)
* Localize Sting messages in every exception.

## Exception Handling

Verbose Error Message (Improper Error Handling) - [Improper Error Handling](https://owasp.org/www-community/Improper_Error_Handling)

Error Message Containing Sensitive Information

Handle Asynchronous exceptions

[Throw; and throw ex;](https://stackoverflow.com/questions/730250/is-there-a-difference-between-throw-and-throw-ex)throw the same object, but its stack trace is modified in different way

* Throw preserves the stack trace. So let’s say Source1 throws Error1 , its caught by Source2 and Source2 says throw then Source1 Error + Source2 Error will be available in the stack trace.
* Throw ex does not preserve the stack trace. So all errors of Source1 will be wiped out and only Source2 error will sent to the client

<https://docs.abp.io/en/abp/latest/Exception-Handling>

* Automatically **handles all exceptions** and sends a standard **formatted error message** to the client for an API/AJAX request.
  + Automatically hides **internal infrastructure errors** and returns a standard error message.
* Provides an easy and configurable way to **localize** exception messages.
* Automatically maps standard exceptions to **HTTP status codes** and provides a configurable option to map custom exceptions.

[**https://stackify.com/csharp-exception-handling-best-practices/**](https://stackify.com/csharp-exception-handling-best-practices/)

**The Basic “try catch finally” Block**

[C# Exceptions and Exception Handling](https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/exceptions/)

[Handle errors in ASP.NET Core](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/error-handling?view=aspnetcore-6.0)

* [.NET 5.0 - Global Error Handler Tutorial](https://jasonwatmore.com/post/2021/05/29/net-5-global-error-handler-tutorial)

API layer

* Global exception handling
  + [Global exception handling VS Try catch everywhere](https://stackoverflow.com/questions/58361013/global-exception-handling-vs-try-catch-everywhere)
    - <https://www.raymondcamden.com/2011/05/26/When-is-it-proper-to-trycatch-versus-global-exception-handling>
  + Standard formatted message or generic message like **“Error occurred, Contact Admin**”
    - [Exception Handling](https://docs.abp.io/en/abp/latest/Exception-Handling#exception-handling), [Consistent error responses](https://medium.com/swlh/clean-architecture-best-exception-handling-with-consistent-responses-in-asp-net-core-api-b22b07a08e38), Error Message Format
* Exception logger

BAL

* Use throw keyword
  + ["throw" and "throw ex" in C#](https://www.c-sharpcorner.com/blogs/difference-between-throw-and-throw-ex-in-c-sharp1)
  + [throw vs. throw(ex)](https://www.dotnetjalps.com/2013/10/throw-vs-throw-ex-csharp.html)
* ArgumentException – Invalid parameters are passed
  + ArgumentNullException

DAL

Handle Specific DB exceptions & SQL Exception

* DbEntityValidationException
* DBException
* SQLException 🡪 SQL ServerDB
  + Catch (Exception e) {If(e.GetType() == typeof(SQLException))}

## Validations

* [Input Validation on Client-Side or Server-Side?](https://www.packetlabs.net/posts/input-validation/) | [JavaScript: client-side vs. server-side validation](https://stackoverflow.com/questions/162159/javascript-client-side-vs-server-side-validation)
  + Always validate on the server – that is the thumb rule
  + Validation on the client is nice for users, but is utterly insecure.
  + User Generic error response model
  + Refer to [FluentValidation](https://docs.fluentvalidation.net/en/latest/) | Chaining validators, Complex Properties, Custom Validators, Custom Validators.
  + API Architecture - Validation Pattern?

HTTP response types

* 400 Bad Request
* [422 Unprocessable Entity](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/422) - [Case study: GitHub API](https://stackoverflow.com/a/52363900)
  + server cannot understand the request due to issues like a semantic error in the request body.
  + server understood the content type of the request entity, and the syntax of the request entity was correct, but it was unable to process the contained instructions
* 401 for Unauthorized requests, when a request requires authentication but it isn't provided
* 403 for Forbidden requests, when a request may be valid but the user doesn't have permissions to perform the action
* 404 for Not found requests, when a resource can't be found to fulfil the request

### Input validation implemented as:

* aka model validation or request validation
  + Validating the format of fields such as email address, phone number, zip code, name, password.
  + Validating mandatory fields
  + Checking the type of data such as string vs number for fields such as social security number.
* An allow list or deny list,
* Validation or Sanitization
* Performed server-side or client-side.
  + Server-Side validation
    - [Model Validation in ASP.NET Web API](https://docs.microsoft.com/en-us/aspnet/web-api/overview/formats-and-model-binding/model-validation-in-aspnet-web-api) 🡪
      * [Data Annotations](https://docs.microsoft.com/en-us/aspnet/web-api/overview/formats-and-model-binding/model-validation-in-aspnet-web-api#data-annotations) 🡪 ModelState
        + [ValidationFilterAttribute](https://code-maze.com/aspnetcore-modelstate-validation-web-api/)
      * [Handling Validation Errors](https://docs.microsoft.com/en-us/aspnet/web-api/overview/formats-and-model-binding/model-validation-in-aspnet-web-api#handling-validation-errors)
      * [FluentValidation](https://docs.fluentvalidation.net/en/latest/) for request validation. And returns BadRequest if fails –
        + FluentValidation ModelState.IsValid
        + [Transforming Values](https://docs.fluentvalidation.net/en/latest/transform.html), Dependent Rules, Inheritance Validation.
    - [EF6 Data Validation](https://docs.microsoft.com/en-us/ef/ef6/saving/validation) using **Data Annotation** or **Fluent API** Or **IValidatableObject**
    - Send only required number of properties in request and response.
* API
  + [How to Implement Input Validation for APIs](https://nordicapis.com/how-to-implement-input-validation-for-apis/)

### Data validations - What data should be validated?

* Ensuring that the value entered is a valid value such as country, date, and so on. Data provide is valid

### Security Input validation

[OWASP Proactive Controls](https://owasp.org/www-project-proactive-controls/) / [2018 Here](https://owasp-top-10-proactive-controls-2018.readthedocs.io/en/latest/index.html)  🡪 [C5: Validate All Inputs](https://owasp.org/www-project-proactive-controls/v3/en/c5-validate-inputs)

[Input Validation Cheat Sheet](https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html) | [API Security Best Practices : Part 4/6 — Input validation](https://medium.com/@hassene/how-to-secure-your-api-part-4-6-input-validation-best-practices-db2c28d7a991)

***Input Validation should not be used as the primary method of preventing***[***XSS***](https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html)***,***[***SQL Injection***](https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html)***and other attacks***

Validate inputs for Cross Site Scripting (XSS), HTML/SQL injection, XEE, SSRF etc.

* HTML injection mitigation- HTML to deface the page.
  + Validating and Sanitizing HTML/HTML Sanitize | [Prevent Cross-Site Scripting (XSS) in ASP.NET Core](https://docs.microsoft.com/en-us/aspnet/core/security/cross-site-scripting?view=aspnetcore-6.0)
* XEE, Server Side Request Forgery (SSRF), XSS

## Constants

[What's the best way to store a group of constants that my program uses? [closed]](https://stackoverflow.com/questions/1724025/whats-the-best-way-to-store-a-group-of-constants-that-my-program-uses)

[Constants class](https://softwareengineering.stackexchange.com/questions/230410/suggest-a-best-practice-to-create-constants-class)

* [**const**](https://msdn.microsoft.com/en-us/library/e6w8fe1b.aspx) keyword: Compile time
* Use static [**readonly**](https://www.c-sharpcorner.com/UploadFile/0c1bb2/read-only-and-constant-in-C-Sharp/): runtime -  settings from the config file
* static property without set
* [[Flags] Enum Attribute](https://stackoverflow.com/questions/8447/what-does-the-flags-enum-attribute-mean-in-c)

|  |
| --- |
| public static class Constants  {  //Assign value at compile time  public **const string** GetCustomerSP = "GetCustomerSP";  //Assigns value at run-time  public **static readonly string** getEnvBaseURL = "<<get value from appsettings.json>>";  //checks where it is android or ios  public **static string** RestMobileUrl = DeviceInfo.Platform == DevicePlatform.Android ? "http://10.0.2.2:5000/api/todoitems/{0}" : "http://localhost:5000/api/todoitems/{0}";  } |
|  |

## Health Checks

[Health checks in ASP.NET Core](https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/health-checks?view=aspnetcore-6.0) 🡪 [Health monitoring](https://docs.microsoft.com/en-us/dotnet/architecture/microservices/implement-resilient-applications/monitor-app-health)

**Health Checks for our microservice**

* Data Stores – SQL, NoSQL, redis, EventStore, Elasticsearch etc.
  + Entity Framework or Dapper
    - Microsoft.Extensions.Diagnostics.HealthChecks.EntityFrameworkCore
* Network, File system or cloud storage – Network Shared Folder, FTP/FTPS, SFTP, SMTP etc.
* Cache server – Redis etc.
* MessageQueues – RabbitMQ, Kafka or Azure Service bus
* OAuth2/OpenId system – Identity Server4
* External URLs or third-party APIs 🡪 [ApiHealthCheck.cs](https://medium.com/it-dead-inside/implementing-health-checks-for-asp-net-core-a-deep-dive-85a327be9a75) | [Customizing the Checks](https://www.telerik.com/blogs/checking-health-aspnet-core-apis#customizing-the-checks)
  + AspNetCore.HealthChecks.Uris
  + Webhooks
* Containers – Docker or kubes
* [Xabaril](https://github.com/Xabaril)/[AspNetCore.Diagnostics.HealthChecks](https://github.com/Xabaril/AspNetCore.Diagnostics.HealthChecks)

**Enhancing Health Checks UI**

* AspNetCore.HealthChecks.UI
* AspNetCore.HealthChecks.UI.Client
* AspNetCore.HealthChecks.UI.InMemory.Storage

Code

|  |
| --- |
| Services.AddHealthChecks();  app.MapHealthChecks("/health");  Services.AddHealthChecksUI().AddInMemoryStorage();  app.MapHealthChecksUI(); |

/health endpoint

/healthchecks-ui endpoint

Links:

* [Health Checks in ASP.NET Core](https://code-maze.com/health-checks-aspnetcore/)
* [Healthchecks in ASP.NET Core – Detailed Guide](https://codewithmukesh.com/blog/healthchecks-in-aspnet-core-explained/)
* [Implementing Health Checks for ASP.NET Core: A deep dive](https://medium.com/it-dead-inside/implementing-health-checks-for-asp-net-core-a-deep-dive-85a327be9a75)
* [Health Monitoring In ASP.NET Core](https://www.c-sharpcorner.com/article/health-monitoring-in-asp-net-core/)

## Test

[NUnit vs. XUnit vs. MSTest: Comparing Unit Testing Frameworks In C#](https://www.lambdatest.com/blog/nunit-vs-xunit-vs-mstest/)

[Unit test](https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/testing?view=aspnetcore-6.0) and [Integration tests in ASP.NET Core](https://docs.microsoft.com/en-us/aspnet/core/test/integration-tests?view=aspnetcore-6.0)

## Async Method

[Building fully Asynchronous ASP.NET Core Web API](https://mithunvp.com/fully-asynchronous-aspnet-core-2-web-api/)

[Asynchronous Programming Best Practices in C#](https://aaronluna.dev/blog/parallel-async-csharp-best-practices-tpl/)

[Long Story Short: Async/Await Best Practices in .NET](https://medium.com/@deep_blue_day/long-story-short-async-await-best-practices-in-net-1f39d7d84050)

### Callbacks

### Angular Callback

Using promise

[JavaScript Async/Await with Angular 7/8 Observable and HTTP Example](https://www.techiediaries.com/javascript-async-await-tutorial/)

[How to use Axios interceptors to poll for long running API calls](https://endjin.com/blog/2020/10/how-to-use-axios-interceptors-to-poll-long-running-api-calls)