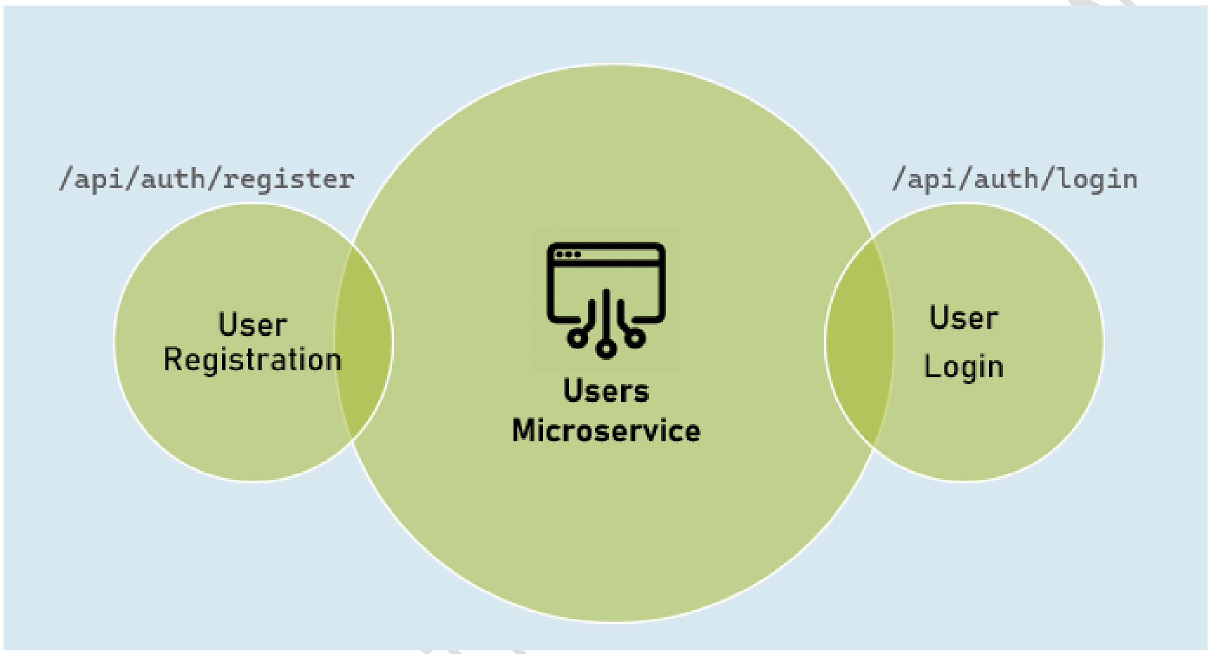
.NET Core Microservices – True Ultimate Guide

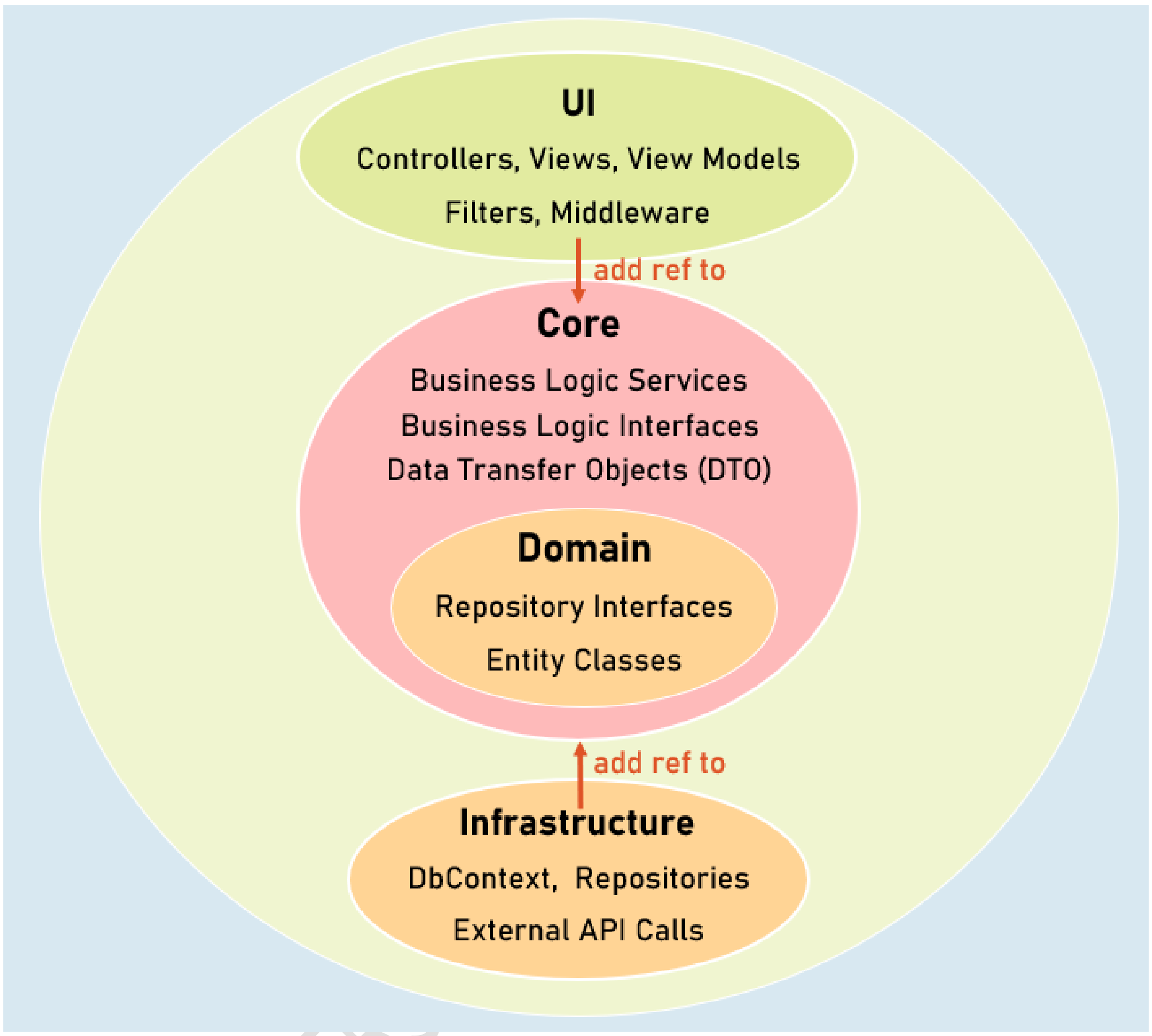
Section 2 – Users Microservice – Cheat Sheet

Creating First Microservice: Users Microservice

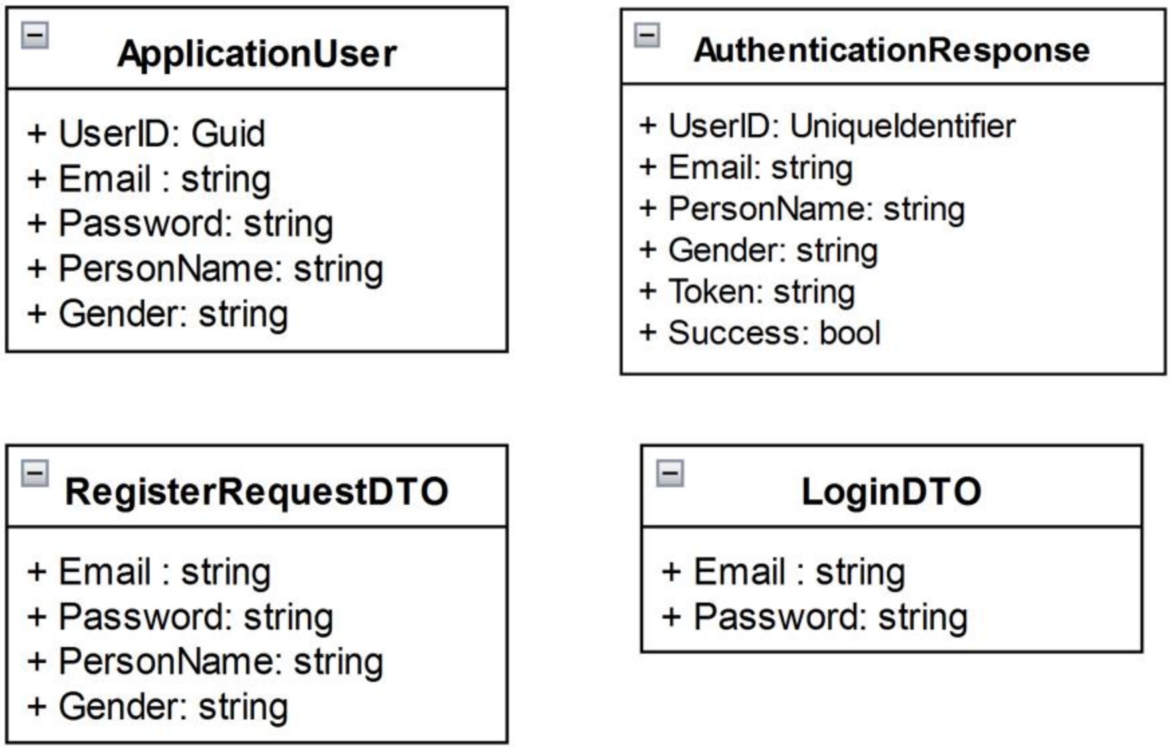
Users Microservice



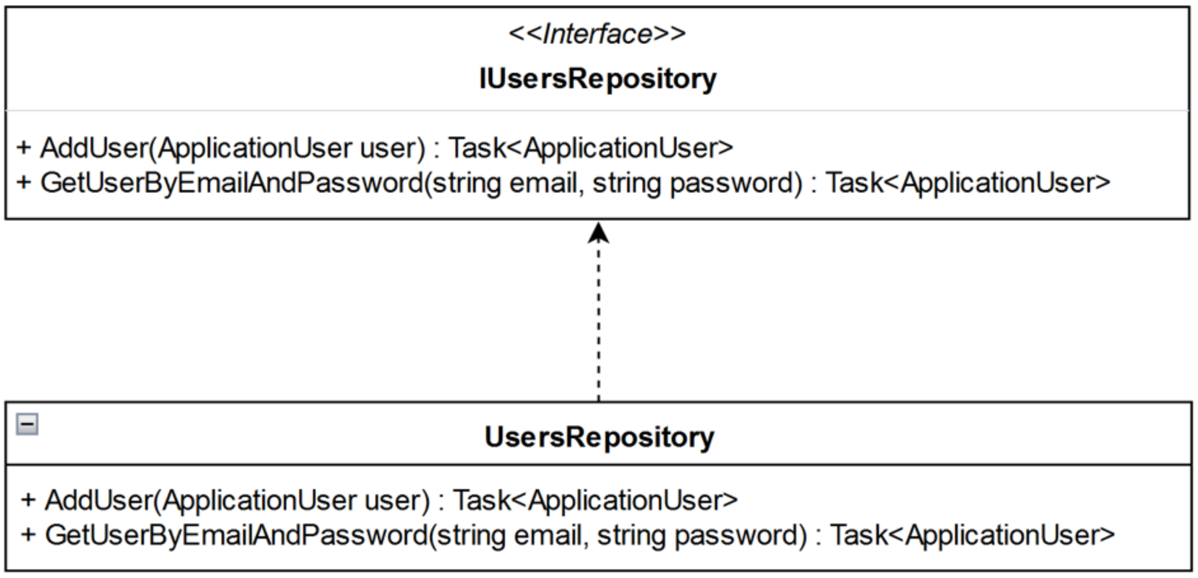
Clean Architecture



User Models

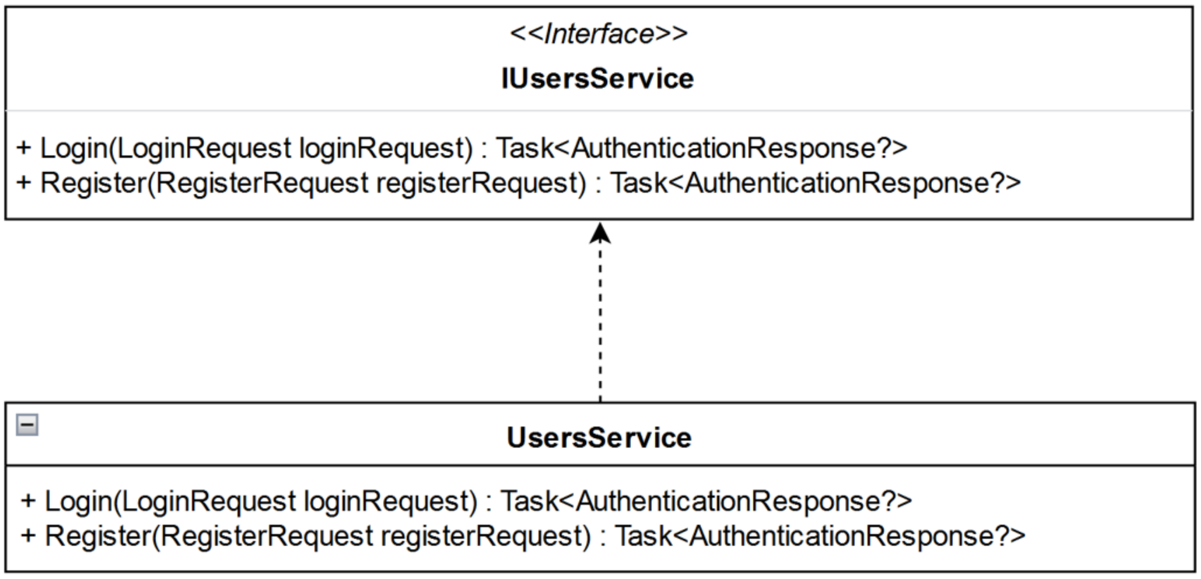

ApplicationUser
+ UserID: Guid
+ Email : string
+ Password: string
+ PersonName: string
+ Gender: string
AuthenticationResponse
+ UserID: UniqueIdentifier
+ Email: string
+ PersonName: string
+ Gender: string
+ Token: string
+ Success: bool
RegisterRequestDTO
+ Email : string
+ Password: string
+ PersonName: string
+ Gender: string
LoginDTO
+ Email : string
+ Password: string

Users Repository



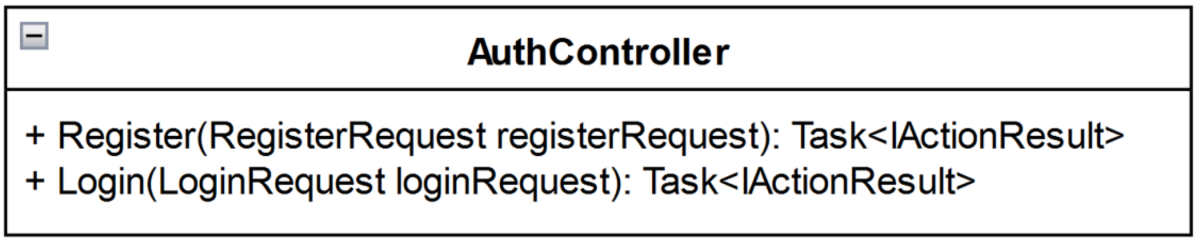
<<Interface>>
IUsersRepository
+ AddUser(ApplicationUser user) : Task<ApplicationUser>
+ GetUserByEmailAndPassword(string email, string password) : Task<ApplicationUser>
UsersRepository
+ AddUser(ApplicationUser user) : Task<ApplicationUser>
+ GetUserByEmailAndPassword(string email, string password) : Task<ApplicationUser>

Users Service



<<Interface>>
IUsersService
+ Login(LoginRequest loginRequest) : Task<AuthenticationResponse?>
+ Register(RegisterRequest registerRequest) : Task<AuthenticationResponse?>
UsersService
+ Login(LoginRequest loginRequest) : Task<AuthenticationResponse?>
+ Register(RegisterRequest registerRequest) : Task<AuthenticationResponse?>

Users Controller



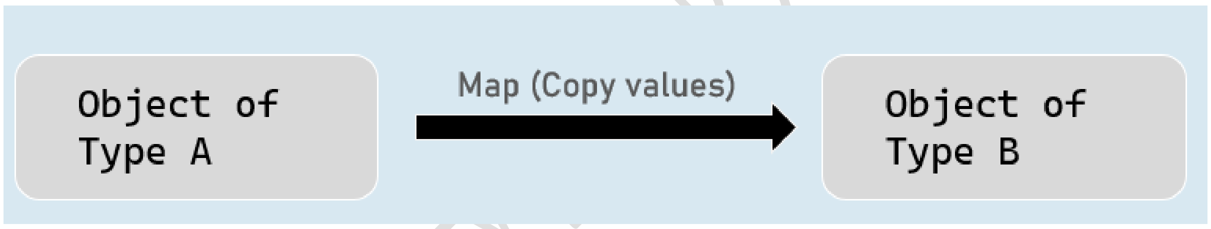
AuthController

+ Register(RegisterRequest registerRequest): Task<IActionResult>
+ Login(LoginRequest loginRequest): Task<IActionResult>

Auto Mapper

AutoMapper is a popular open-source library that simplifies the process of mapping data between objects of different types.

Instead of writing repetitive code to manually copy values from one object to another, AutoMapper allows you to define mapping configurations in a clear and concise way, reducing the amount of code you need to write and maintain.



A) Without AutoMapper

//Assume, source object already exists.

//Create destination object

DestinationClass destinationVariable = new DestinationClass();

//copy each property value from source object to destination object

destinationVariable.Property1 = sourceVariable.Property1;

destinationVariable.Property2 = sourceVariable.Property2;

…

B) With AutoMapper

//Assume, source object already exists.

//Create destination object and copy data from source object

DestinationClass destinationVariable = mapper.Map<DestinationClass>(sourceVariable);

NuGet Packages of AutoMapper

AutoMapper

This package is the core AutoMapper library that provides the fundamental functionality for object-to-object mapping.

It includes the "Mapper" class (using which you can perform object mappings), configuration mechanisms and the core mapping logic.

AutoMapper.Extensions.Microsoft.DependencyInjection

This package is an extension to the core "AutoMapper" package.

It provides extension methods allows us to add auto mapper profiles as services into the IoC container.

PostgreSQL

PostgreSQL (or Postgres) is an open-source, cross-platform relational DBMS, designed to handle high range of workloads.

- Cost affective (no licensing cost)
- Cross-platform (supports Linux hosting also)
- Built-in support for JSON & NoSQL

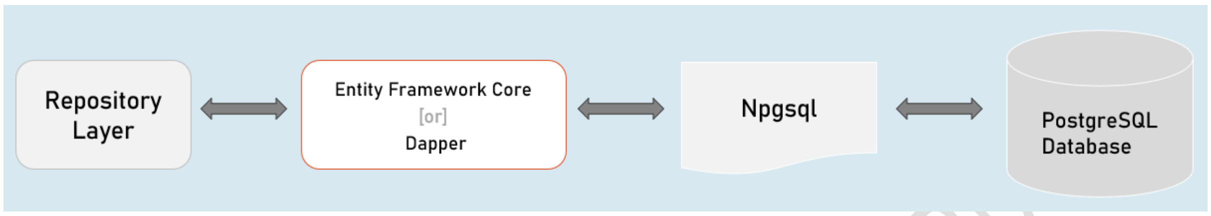
NuGet Packages of Postgres

Npgsql

This package is the Managed Data Provider for PostgreSQL, allowing .NET applications to connect and interact with PostgreSQL databases.

Npgsql stands for ".NET PostgreSQL".

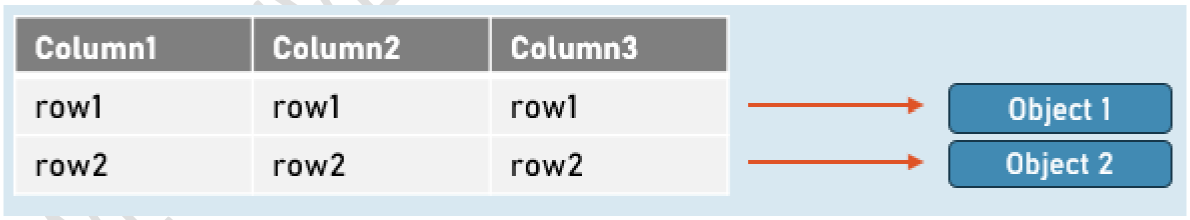
It supports both "Entity Framework Core" and "Dapper".



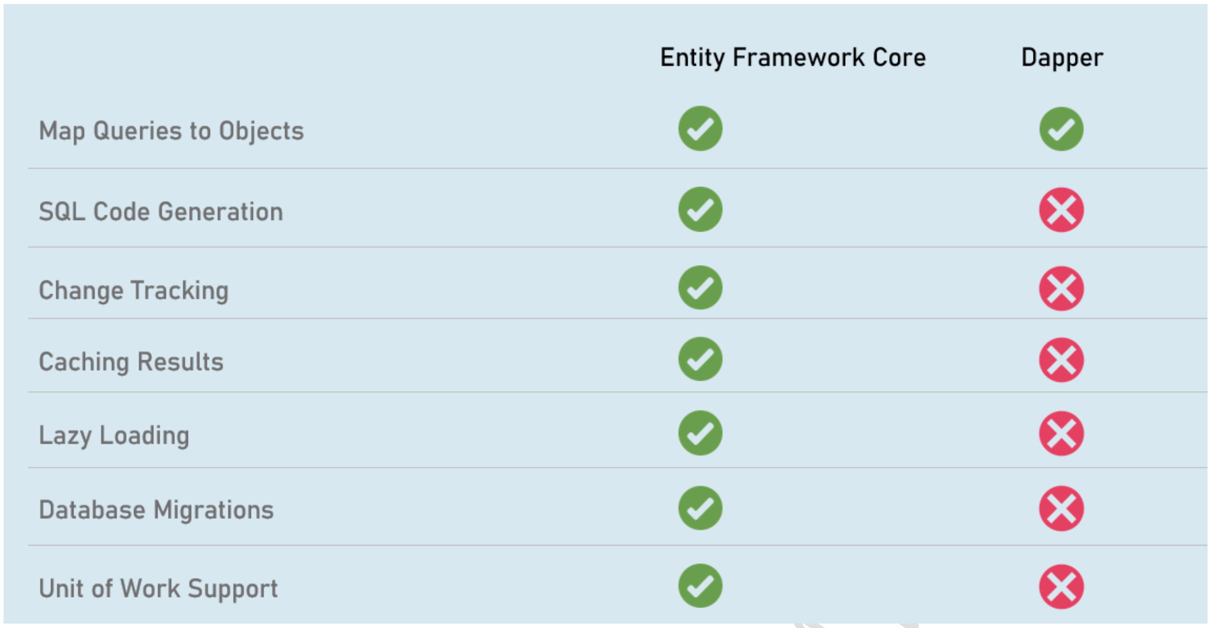
Dapper

Dapper is an open-source, light-weight, high-performance ORM (Object-Relational Mapping) library for .NET.

- Faster than EF Core in most of the scenarios.
- Provides SQL Injection attacks by default.
- Provides grip on raw SQL queries.
- Allows you to map query results to strongly-typed objects.



Features List

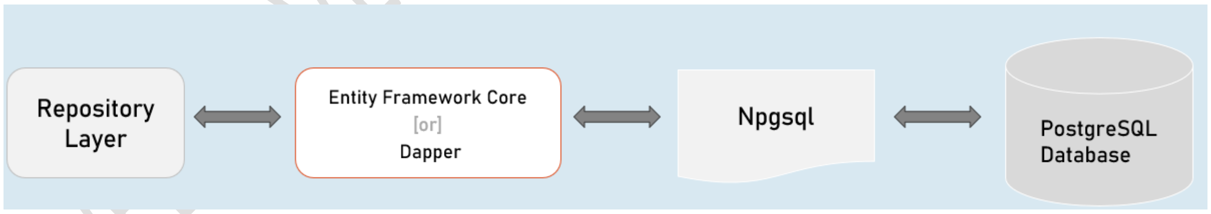


	Entity Framework Core	Dapper
Map Queries to Objects	✓	✓
SQL Code Generation	✓	✗
Change Tracking	✓	✗
Caching Results	✓	✗
Lazy Loading	✓	✗
Database Migrations	✓	✗
Unit of Work Support	✓	✗

NuGet Packages of Dapper

Dapper

This package encapsulates the Dapper framework's essential components and functionalities, which allows the developers to execute SQL scripts against databases such as SQL Server, MySQL or PostgreSQL etc.



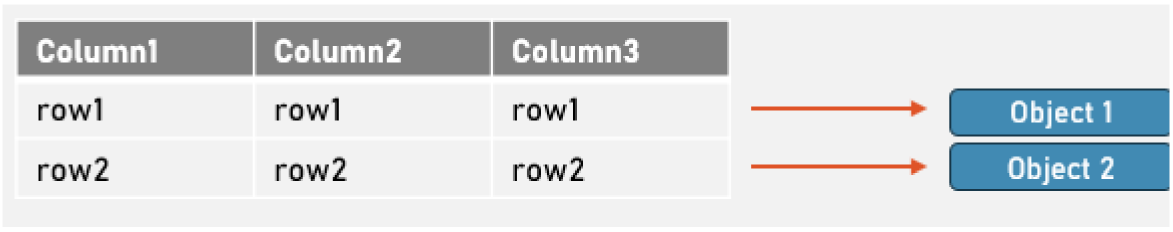
Dapper Methods

1. QueryAsync<T>

Executes a SQL query (SELECT statement) and maps the results to a strongly-typed list of objects.

IEnumerable<MyClass> result = await connection.QueryAsync<MyClass>(

```
"SELECT * FROM TableName");
```



2. QueryFirstOrDefaultAsync<T>

Executes a SQL query (SELECT statement), reads the first row of the result set and maps the results to a strongly-typed list of objects.

It returns the default value (null) if no rows selected.

```
MyClass? result = await connection.QueryFirstOrDefaultAsync<MyClass>(

"SELECT * FROM TableName WHERE Condition");
```

3. ExecuteAsync<T>

Executes a SQL command (INSERT, UPDATE or DELETE statements) and returns the number of affected rows.

```
int rowsCount = await connection.ExecuteAsync(

"INSERT INTO TableName(Column1, Column2) VALUES(@Property1, @Property2)", modelObject);
```

```
int rowsCount = await connection.ExecuteAsync(

"UPDATE TableName SET Column1 = @Property1 WHERE Condition", modelObject);
```

```
int rowsCount = await connection.ExecuteAsync(

"DELETE FROM TableName WHERE Condition");
```

Fluent Validation

FluentValidation is a popular .NET library that provides a fluent mechanism for defining model validations.

Data Annotations

```
public class ModelClass
{
  [Required(ErrorMessage = "Required Error Message")]
  [StringLength(maxValue, MinimumLength = minValue, ErrorMessage = "Length Error Message")]
  public type Property1 { get; set; }
}
```

Fluent Validation

```
public class Validator : AbstractValidator<ModelClass>
{
  public PersonValidator()
  {
    RuleFor(model => model.Property1)
      .NotEmpty().WithMessage("Required Error Message")
      .Length(minValue, maxValue).WithMessage("Length Error Message");
  }
}
```

Commonly used Fluent Validators

1. NotNull( )

Ensures that a property value is not null.

e.g: RuleFor(model => model.PropertyName).NotNull();

Applicable on: Any nullable type

2. Null( )

Ensures that a property value is null.

e.g: RuleFor(model => model.PropertyName).Null();

Applicable on: Any nullable type

3. NotEmpty( )

Ensures that a property value is not empty string, null, whitespace, default value for value types (e.g: 0 for int) or empty collection.

e.g: RuleFor(model => model.PropertyName).NotEmpty();

Applicable on: Any type

4. Empty( )

Ensures that a property value is empty string, null, whitespace, default value for value types (e.g: 0 for int) or empty collection.

e.g: RuleFor(model => model.PropertyName).Empty();

Applicable on: Any type

5. MaximumLength( )

Ensures that the length of the property value is less than or equal to the specified value.

e.g: RuleFor(model => model.PropertyName).MaximumLength(maximum);

Applicable on: string

6. MinimumLength( )

Ensures that the length of the property value is greater than or equal to the specified value.

e.g: RuleFor(model => model.PropertyName).MinimumLength(minimum);

Applicable on: string

7. Length( )

Ensures that the length of the property value is greater than or equal to the specified minimum value and less than or equal to the specified maximum value.

e.g: RuleFor(model => model.PropertyName).Length(minimum, maximum);

Applicable on: string

8. InclusiveBetween( )

Ensures that a property value is greater than or equal to the specified minimum value and less than or equal to the specified maximum value.

e.g: RuleFor(model => model.PropertyName).InclusiveBetween(minimum, maximum);

Applicable on: numeric types

9. ExclusiveBetween( )

Ensures that a property value is greater than to the specified minimum value and less than to the specified maximum value.

e.g: RuleFor(model => model.PropertyName).ExclusiveBetween(minimum, maximum);

Applicable on: numeric types

10. Equal( )

Ensures that a property value is equal to the specified value.

e.g: RuleFor(model => model.Property1).Equal(model => model.Property2);

e.g: RuleFor(model => model.PropertyName).Equal(value);

Applicable on: Any type

11. NotEqual( )

Ensures that a property value is not equal to the specified value.

e.g 1: RuleFor(model => model.Property1).NotEqual(model => model.Property2);

e.g 2: RuleFor(model => model.PropertyName).NotEqual(value);

Applicable on: Any type

12. Matches( )

Ensures that a property value matches with the specified regular expression.

e.g: RuleFor(model => model.PropertyName).Matches("regular expression");

Applicable on: string

13. EmailAddress( )

Ensures that a property value matches with the valid email address format.

e.g: RuleFor(model => model.PropertyName).EmailAddress();

Applicable on: string

14. IsInEnum( )

Ensures that a property value is a valid value of that enum.

e.g: RuleFor(model => model.PropertyName).IsInEnum();

Applicable on: enum types

15. Custom Validator

Ensures that a property value matches with the specified conditional expression.

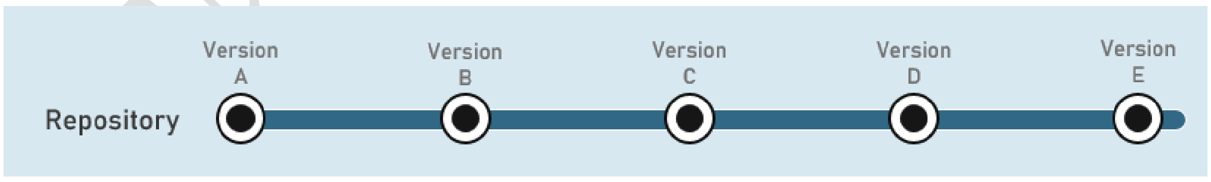
e.g: RuleFor(model => model.PropertyName).Must(val => CustomMethod(val));

Applicable on: Any types

Git

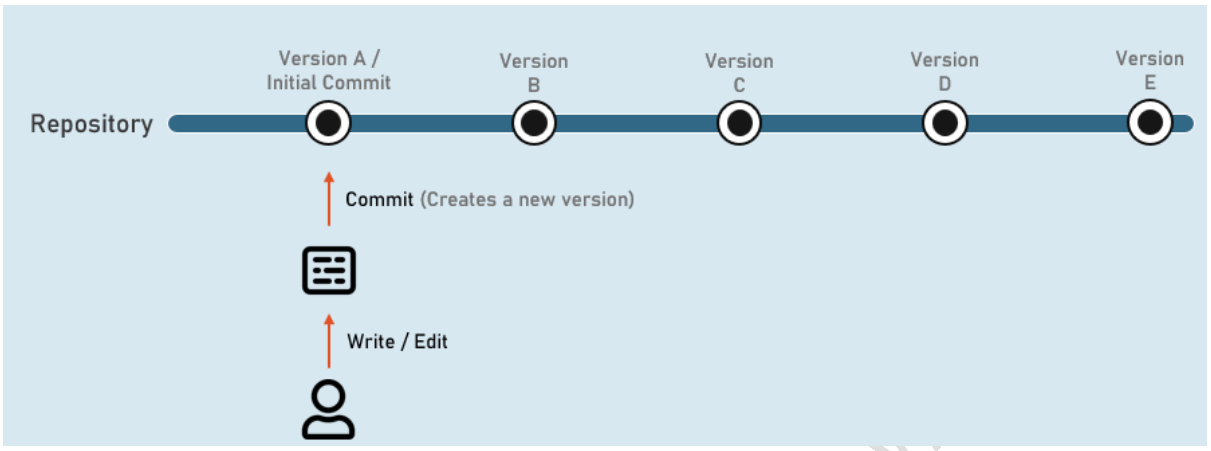
Git is a distributed version control system for tracking changes in source code during software development.

A version control system (VCS) allows multiple developers to contribute to the source code efficiently without needing to interfere with other's work.



Git Repository

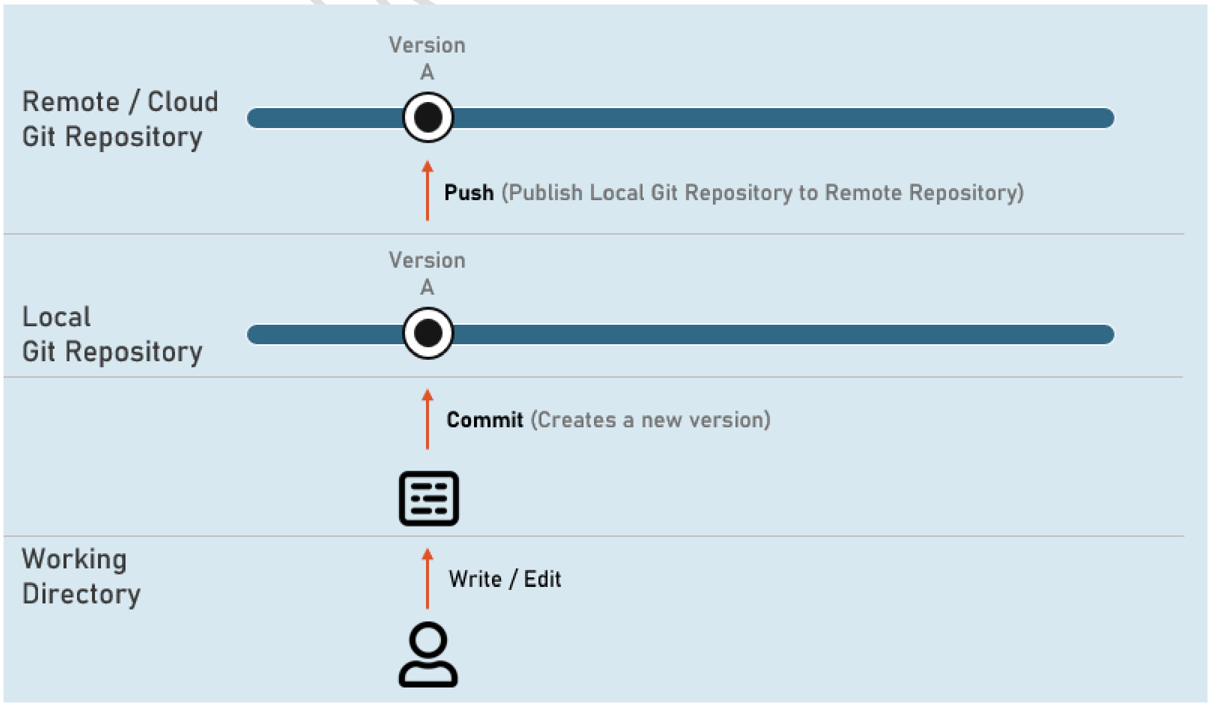
Git repository is a data structure that stores complete history of source code changes that includes one or more branches and their versions.



A commit / version represents a state of the Git repository at a specific point of time that includes metadata (including author, commit message, timestamp), Unique identifier (SHA-1 Hash) and Tree Object (includes only changes made to the files; but not entire copy of the files).

GitHub

GitHub is a cloud-based hosting platform that enables developers to host Git repositories (as private or public repositories) on remote servers.


Remote / Cloud Git Repository
Version A
Push (Publish Local Git Repository to Remote Repository)
Local Git Repository
Version A
Commit (Creates a new version)
Working Directory
Write / Edit

Swagger

Swagger is a set of open-source tools that help developers to generate interactive UI to document, test RESTful services.

Swagger is a set of tools to implement Open API.

1. Swasbuckle.AspNetCore

(Framework that makes it easy to use swagger in asp.net core)

2. Swagger

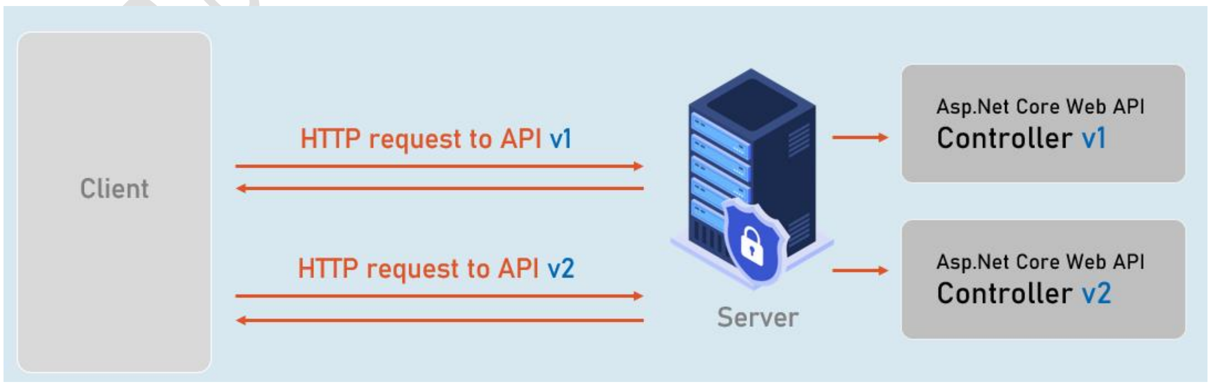
(Set of tools to generate UI to document & test RESTful services)

3. Open API

(Specification that defines how to write API specifications in JSON)

Swagger API Versions

API Versioning is the practice of transparently managing changes to your API, where the client requests a specific version of API; and the server executes the same version of the API code.


Client
HTTP request to API v1
HTTP request to API v2
Server
Asp.Net Core Web API
Controller v1
Asp.Net Core Web API
Controller v2