

CQRS/MEDIATR ET MINIMAL API

UNE APPROCHE PRAGMATIQUE



N'oublier pas d'évaluer ma présentation !
Vos commentaires sont le bienvenu.



Qui suis-je ?

Vie personnelle

- 👨👩 Fier papa de deux jeunes enfants
- 👨👩👧 Famille d'accueil avec la DPJ
- 🇨🇦 Fier partisan des Canadiens de Montréal... Go Habs Go !

Parcours professionnel

- 🏆 Baccalauréat en génie logiciel, certifié PSPO 1
- 🔥 Tech Lead chez Openmind (7 ans)
- 😊 Plus de 15 ans d'expérience en développement web (PHP, dotNet)
- 🎯 Rôles: Développeur → PO → Project Lead → Team Lead → Tech Lead

Mes valeurs

- 🌟 Passionné par l'amélioration continue et l'innovation
- 🌟 Construire des équipes fortes par la collaboration et l'empathie



erik-beaulieu



matrix818181



Qui sommes-nous ?

Openmind Technologies : votre partenaire technologique pour croître !

- > Fondée en 2005, une entreprise de premier plan spécialisée dans le **développement de logiciels et d'applications sur mesure**.
- > Experts en **automatisation des processus** pour les PME des secteurs **de la fabrication, de la distribution et de la construction**.
- > Plus de 50 employés et experts à votre service.



Rejoignez-nous dans l'aventure !



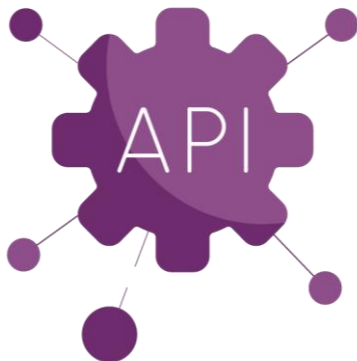
Nos distinctions



Notre expertise

Experts en automatisation numérique, innovation et modernisation des logiciels, au-delà des solutions existantes.

Automatiser.



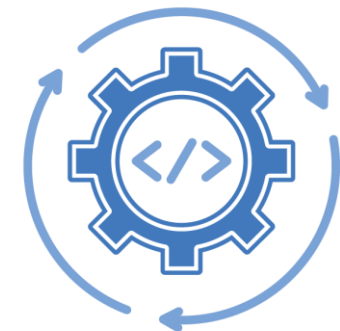
**Pont de données,
intergiciel et API**

Innover.



**Développement de
produits numériques**

Moderniser.



Modernisation des logiciels



< COMPRENDRE D'OÙ ON ARRIVE
POUR SAVOIR OÙ ON S'EN VA />



Nos premiers pas...

Technologie: dotNet Core 2.2

Architecture: N-Tier avec controllers, services et repositories

Connaissances: Très peu, base éducative

> Avantages

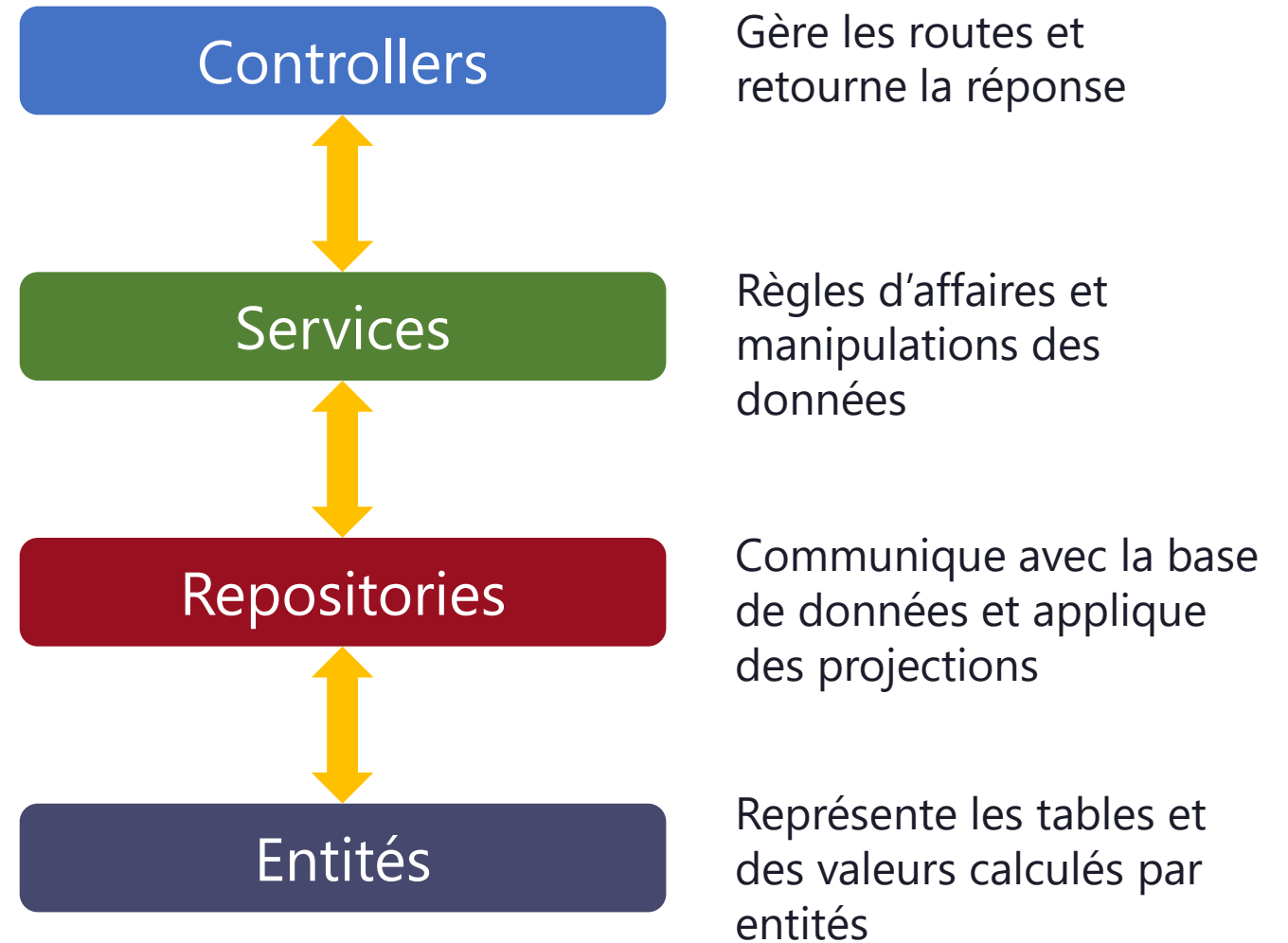
- Simple d'approche
- Aucune projection d'entités (garbage-in/garbage-out)
- Gestion des logiques d'affaire dans les services

> Inconvénients

- Comprendre la séparation entre la donnée (entité) et la présentation de la donnée (DTO)
- Beaucoup de relations entre les entités, beaucoup de données générées par l'API
- Références circulaires entre les services
- Difficile à modifier avec le temps



Architecture N-Tier





Apprendre à marcher...

Technologie: dotNet 5

Architecture: Clean-Code

Connaissances: Intermédiaire

> Avantages

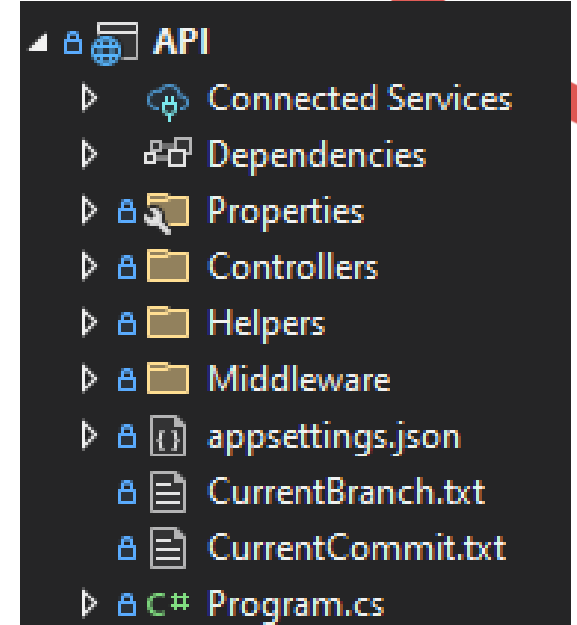
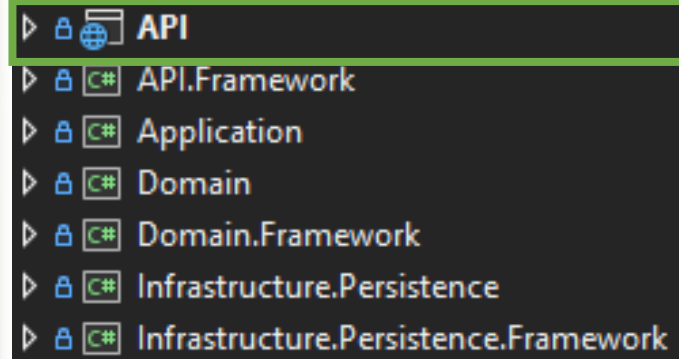
- Moins de code inutile
- Code plus modulaire
- Évolutif et adaptable

> Inconvénients


- Complexe à comprendre
- Demande plus d'effort pour comprendre où mettre le code
- Nécessite plus de code (interfaces + services) afin de faire une opération simple
- Avec une équipe moins expérimentée, le code ne finit jamais à la bonne place



Clean-code



Clean-code


A child in a grey sweater and blue pants is walking on a wooden floor with a large, colorful rainbow rug. In the background, there is a light blue sofa with purple and green cushions.

```
API
API.Framework
Application
Domain
Domain.Framework
Infrastructure.Persistence
Infrastructure.Persistence.Framework
```

```
Application
  Dependencies
  Properties
  BackgroundTasks
  Behaviors
  Commands
  Constants
  Features
  Handlers
  Helpers
  Models
  Queries
  ReportBuilders
  Resources
  Responses
  Services
  Validations
  Validators
  AppModule.cs
  BackgroundTaskModule.cs
  TaskSchedulerModule.cs
```



Clean-code


A child in a grey sweater and blue pants is walking on a wooden floor with a large, colorful rainbow rug. In the background, there is a light blue sofa with purple and green cushions.

```
API
API.Framework
Application
Domain
Domain.Framework
Infrastructure.Persistence
Infrastructure.Persistence.Framework
```

```
Domain
  Dependencies
  Abstracts
  Constants
  DbFunctions
  Entities
  Exceptions
  Extensions
  Helpers
  Interfaces
  Views
  C# CurrentUserProvider.cs
```




Clean-code

A child in a grey sweater and blue pants is walking on a wooden floor with a large, colorful rainbow rug. In the background, there is a light blue sofa with green and purple cushions.

```
API
API.Framework
Application
Domain
Domain.Framework
Infrastructure.Persistence
Infrastructure.Persistence.Framework
```

```
Infrastructure.Persistence
├── Dependencies
├── Constants
├── Helpers
├── Migrations
├── Repositories
├── DefaultDbContextProvider.cs
├── PersistenceModule.cs
└── RepositoryContext.cs
```





Comment on
peut faire
mieux ?



Aosome

ringing

Incoming Call

Messages

Voxation





La grande réflexion s'amorce...

Technologie: dotNet 6

Architecture: ?????

> **Ce que je veux**

- Produire des fonctionnalités plus rapidement
- Isolation des logiques d'affaire
- Pouvoir travailler en équipe sur plusieurs fonctionnalités en même temps (blocs Lego)
- Utiliser les Endpoints (nouveau en .NET 6)

> **Ce que je ne veux pas**

- Gérer plusieurs sous-projets
- Perdre du temps à retrouver le code d'une fonctionnalité (controller → service → repository)
- Plusieurs fichiers de constantes





Vertical slicing ?

Technologie: dotNet 6

Architecture: Vertical slicing... ?

> Ce que je veux

- ✓ • Produire des fonctionnalités plus rapidement
- ✓ • Isolation des logiques d'affaire
- ✓ • Pouvoir travailler en équipe sur plusieurs fonctionnalités en même temps (blocs Lego)
- ✓ • Utiliser les Endpoints (nouveau en .NET 6)

> Ce que je ne veux pas

- ✓ • Gérer plusieurs sous-projets
- ✓ • Perdre du temps à retrouver le code d'une fonctionnalité (controller → service → repository)
- ✓ • Plusieurs fichiers de constantes



<CQRS, MEDIATR, ETC...
QU'EST-CE QUE ÇA MANGE EN
HIVER ? />

A person in a dark jacket and trousers stands with their back to the camera at a road junction. The road splits into two paths that curve away from each other. The scene is hazy, with trees and a bright light source in the distance, creating a contemplative atmosphere.

CQRS

Command Query Responsibility Segregation

Séparation logique entre les opérations de lecture et d'écriture pour maintenir des performances accrues.

> **Command (CUD → CRUD – R)**

- Opérations de création, modification et suppression de données
- Objets plus complexes car ils représentent souvent les ressources de l'API qui doivent être mise à jour

> **Query (Lecture)**

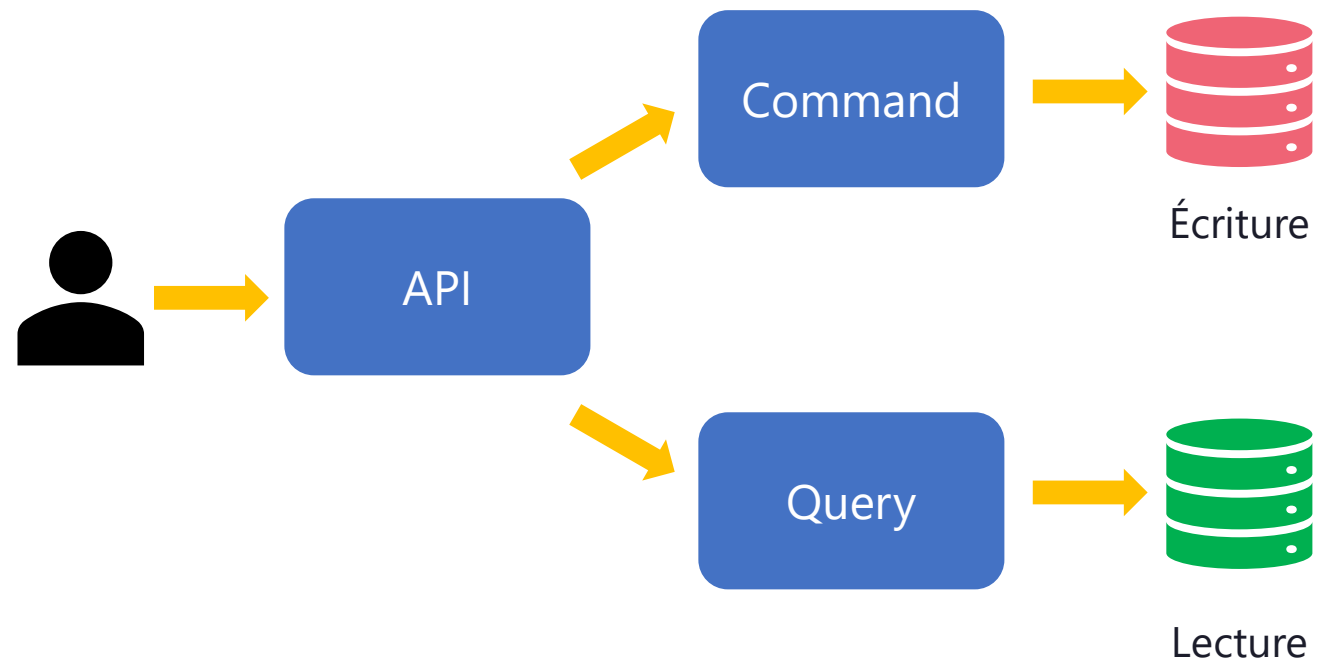
- Opération de lecture
- Possède souvent des paramètres de:
 - Pagination
 - Ordonnancement
 - Filtrage



CQRS

Command Query Responsibility Segregation

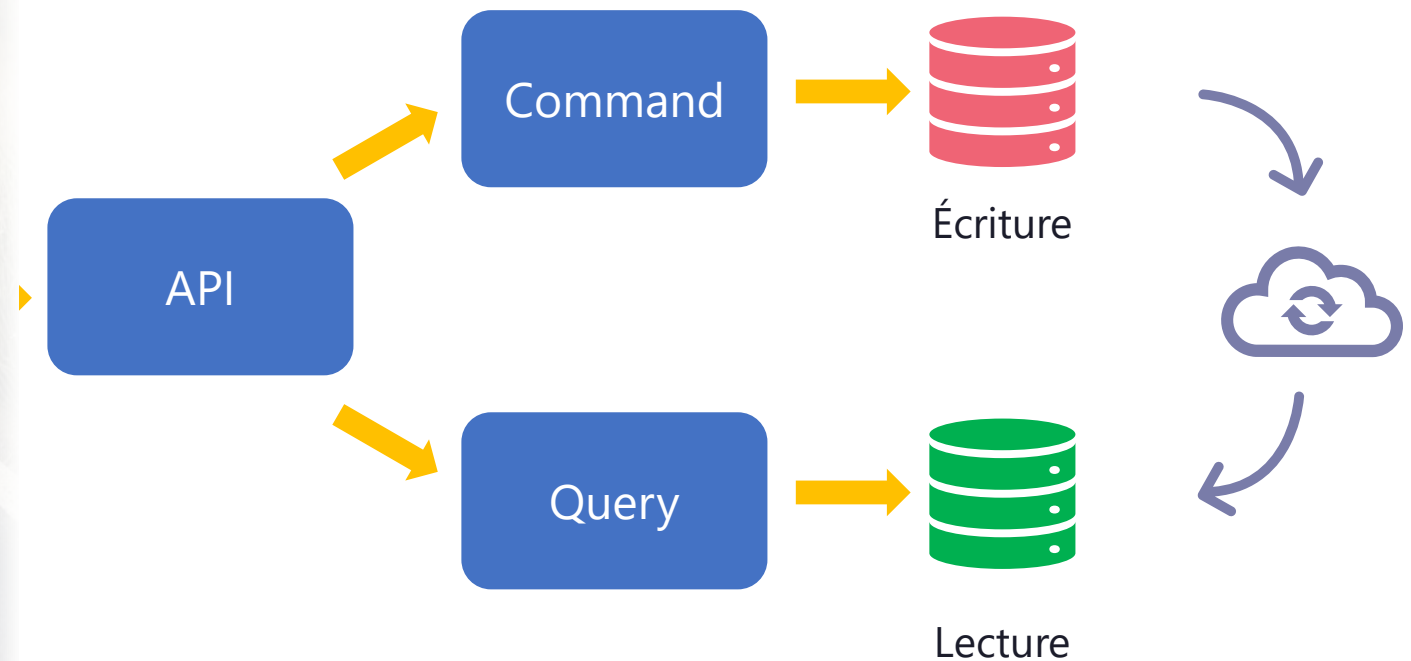
Séparation logique entre les opérations de lecture et d'écriture pour maintenir des performances accrues.



CQRS

Command Query Responsibility Segregation

Séparation logique entre les opérations de lecture et d'écriture pour maintenir des performances accrues.





MediatR

Librairie open-source développée par Jimmy Bogart qui met en œuvre le **patron de conception Médiateur** dans les applications DotNet. De plus, il est aussi possible de l'utiliser sous un format pub-sub.

> Avantages

- Découplage des composants → Les objets communiquent via le médiateur plutôt que directement entre eux.
- Facilite la maintenance → Réduction des dépendances directes = modification plus simple du code.
- Améliore la testabilité → Les objets deviennent plus unitaires et peuvent être testés indépendamment.
- Clarifie les responsabilités → Chaque objet possède un rôle bien défini sans dépendre d'autres.
- Évolutif → Ajout de nouvelles interactions sans modifier les classes existantes.





MediatR

Librairie open-source développée par Jimmy Bogart qui met en œuvre le **patron de conception Médiateur** dans les applications DotNet. De plus, il est aussi possible de l'utiliser sous un format pub-sub.

> Inconvénients

- Ajoute un niveau d'indirection → Peut rendre le débogage plus difficile, car les flux de données ne sont pas toujours explicites.
- Moins naturel sur de petits projets → Introduit une complexité inutile si l'application est simple.





MediatR - Équivalence

Plusieurs bibliothèques suivent le même concept que MediatR dans plusieurs autres langages, voici une liste non exhaustive:

- > Java – Sprint via @EventListener
- > **Typescript – NestJS avec nestjs/cqrs**
- > Python – Pydispatcher
- > Go – Watermill
- > Ruby – Wisper
- > PHP – Laravel Event Bus
- > PHP – Symfony Event Dispatcher





MediatR

Une fonctionnalité est **composée de 3 éléments**, soit

- > **Requête (Command/Query)**
 - Enveloppe dans laquelle on spécifie les paramètres pour le gestionnaire
 - Implémente l'interface *IRequest<TResponse>*
- > **Gestionnaire (Handler)**
 - Effectue le travail souhaité
 - Implémente l'interface *IRequestHandler<TRequest, TResponse>*
- > **Réponse (Response)**
 - Résultat de l'exécution du gestionnaire





MediatR – Pipelines / Behaviors

Permet d'ajouter des comportements avant ou après l'exécution d'une fonctionnalité. On peut les comparer aux middlewares.

Voici quelques exemples pratiques:

- > **Validation (avec FluentValidator)**
 - Permet de valider tous les paramètres d'entrées avant l'exécution de la fonctionnalité
- > **Caching (avec IMemoryCache)**
 - Permet de mettre le résultat d'une fonctionnalité en cache afin de récupérer la valeur rapidement
- > **DatabaseTransaction**
 - Permet d'encapsuler l'exécution d'une fonctionnalité dans une transaction SQL





MediatR – Pipelines / Behaviors

Permet d'ajouter des comportements avant ou après l'exécution d'une fonctionnalité. On peut les comparer aux middlewares.

Voici quelques exemples pratiques:

- > **Logging**

- Permet d'ajouter au *logger* des informations sur l'exécution

- > **Performance**

- Permet de chronométrer l'exécution d'une fonctionnalité



<OK, JE COMPRENDS... !
EST-CE QU'ON PEUT ENFIN VOIR
DU CODE ? />

```
using FluentValidation;
using MediatR;

namespace API.Features.Accesses;
public class Skeleton
{
    public class Response
    {
    }

    public class Endpoint : AbstractEndpoint
    {
        public override void MapEndpoint(IEndpointRouteBuilder app)
        {
        }
    }

    public class Query : IRequest<Response>
    {
    }
}
```



```
}  
}  
  
public class Query : IRequest<Response>  
{  
}  
  
public class Validator : AbstractValidator<Query>  
{  
    public Validator()  
    {  
    }  
}  
  
public class Handler : IRequestHandler<Query, Response>  
{  
    public async Task<Response> Handle(Query request, CancellationToken cancellationToken)  
    {  
        return new Response();  
    }  
}
```

```
public class ValidationBehavior<TRequest, TResponse>(IEnumerable<IValidator<TRequest>> validators) : IPipelineBehavior<TRequest, TResponse>
{
    private readonly IEnumerable<IValidator<TRequest>> _validators = validators;

    public async Task<TResponse> Handle(
        TRequest request,
        RequestHandlerDelegate<TResponse> next,
        CancellationToken cancellationToken
    )
    {
        if (_validators.Any())
        {
            var context = new ValidationContext<TRequest>(request);

            var validationResults = await Task.WhenAll(
                _validators.Select(v => v.ValidateAsync(context, cancellationToken))
            );

            var failures = validationResults
                .SelectMany(r => r.Errors)
                .Where(f => f != null)
                .GroupBy(
                    x => x.PropertyName,
                    x => x.ErrorMessage,
                    (propertyName, errorMessages) => new
                    {
                        Key = propertyName,
                        Values = errorMessages.Distinct().ToArray()
                    })
                .ToDictionary(x => x.Key, x => x.Values);

            if (failures.Count != 0)
```



```
RequestHandlerDelegate<TResponse> next,  
CancellationToken cancellationToken  
)  
{  
    if (_validators.Any())  
    {  
        var context = new ValidationContext<TRequest>(request);  
  
        var validationResults = await Task.WhenAll(  
            _validators.Select(v => v.ValidateAsync(context, cancellationToken))  
        );  
  
        var failures = validationResults  
            .SelectMany(r => r.Errors)  
            .Where(f => f != null)  
            .GroupBy(  
                x => x.PropertyName,  
                x => x.ErrorMessage,  
                (propertyName, errorMessages) => new  
                {  
                    Key = propertyName,  
                    Values = errorMessages.Distinct().ToArray()  
                })  
            .ToDictionary(x => x.Key, x => x.Values);  
  
        if (failures.Count != 0)  
            throw new RestException(System.Net.HttpStatusCode.BadRequest, failures);  
    }  
  
    return await next();  
}
```

```
public class Create
{
    public class Response
    {
        public Guid Id { get; set; }
        public string Slug { get; set; }
        public string Title { get; set; }
        public string Description { get; set; }
        public string Body { get; set; }
    }

    public class Endpoint : AbstractEndpoint
    {
        public override void MapEndpoint(IEndpointRouteBuilder app)
        {
            app
                .MapPost("articles", Handle<Response, Command>)
                .WithName(nameof(Create))
                .WithTags(nameof(Articles))
                ;
        }
    }

    public class Command : IRequest<Response>
    {
        public string Title { get; set; }
        public string Description { get; set; }
        public string Body { get; set; }
        public IEnumerable<string> Tags { get; set; }
    }
}
```




```
public class Command : IRequest<Response>
{
    public string Title { get; set; }
    public string Description { get; set; }
    public string Body { get; set; }
    public IEnumerable<string> Tags { get; set; }
}
```

```
public class Validator : AbstractValidator<Command>
{
    public Validator()
    {
        RuleFor(x => x.Title)
            .NotEmpty()
            .MinimumLength(10)
            .MaximumLength(100)
            .Must((title) => title != "Hello World");

        RuleFor(x => x.Description).NotEmpty();

        RuleFor(x => x.Body).NotEmpty();

        RuleFor(x => x.Tags).NotEmpty();
    }
}
```

```
public class Handler(IRepositoryWithOperators<Article> articleRepository) : IRequestHandler<Command, Response>
{
    public async Task<Response> Handle(Command request, CancellationToken cancellationToken)
```



```
        RuleFor(x => x.Tags).NotEmpty();
    }
}

public class Handler(IRepositoryWithOperators<Article> articleRepository) : IRequestHandler<Command, Response>
{
    public async Task<Response> Handle(Command request, CancellationToken cancellationToken)
    {
        var article = new Article()
        {
            Title = request.Title,
            Description = request.Description,
            Body = request.Body,
            Tags = request.Tags
        };

        await articleRepository.Add(article, cancellationToken);

        return new Response()
        {
            Id = article.Id,
            Slug = article.Slug,
            Title = article.Title,
            Description = article.Description,
            Body = article.Body
        };
    }
}
}
```



```
using MediatR;
using Microsoft.Extensions.Caching.Memory;

namespace API.Behaviors;

public interface ICacheService
{
    Task<T> GetOrCreateAsync<T>(
        string key,
        Func<CancellationToken, Task<T>> factory,
        TimeSpan? expiration = null,
        Cancellation_token cancellationToken = default);

    void Remove(string key);
}

public interface ICacheable
{
    /// <summary> Indicates whether the request should bypass the cache </summary>
    public bool GetBypassCache();

    /// <summary> Custom key for the cached item </summary>
    public string GetCacheKey();

    /// <summary> Sliding expiration duration for the cached item </summary>
    public TimeSpan? GetSlidingExpiration();
}

public class CachingBehavior<TRequest, TResult>(ICacheService cacheService) : IPipelineBehavior<TRequest, TResult> where TRequest : ICacheable
{
    public async Task<TResult> Handle(TRequest request, RequestHandlerDelegate<TResult> next, Cancellation_token cancellationToken)
    {
        if (request.GetBypassCache())
            return await next();
    }
}
```



```

public class CachingBehavior<TRequest, TResult>(ICacheService cacheService) : IPipelineBehavior<TRequest, TResult> where TRequest : ICacheable
{
    public async Task<TResult> Handle(TRequest request, RequestHandlerDelegate<TResult> next, CancellationToken cancellationToken)
    {
        if (request.GetBypassCache())
            return await next();

        return await cacheService.GetOrCreateAsync(
            request.GetCacheKey(),
            _ => next(),
            request.GetSlidingExpiration(),
            cancellationToken
        );
    }
}

```

```

internal sealed class CacheService(IMemoryCache cache) : ICacheService
{
    private static readonly TimeSpan DefaultExpiration = TimeSpan.FromMinutes(10);
    private readonly IList<string> _keys = [];

    public async Task<T> GetOrCreateAsync<T>(
        string key,
        Func<CancellationToken, Task<T>> factory,
        TimeSpan? expiration = null,
        CancellationToken cancellationToken = default
    )
    {
        T result = await cache.GetOrCreateAsync(
            key,
            entry =>
            {
                entry.SetSlidingExpiration(expiration ?? DefaultExpiration);
                return factory(cancellationToken);
            }
        );
    }
}

```

```
    T result = await cache.GetOrCreateAsync(
        key,
        entry =>
        {
            entry.SetSlidingExpiration(expiration ?? DefaultExpiration);
            return factory(cancellationToken);
        }
    );

    if (!_keys.Contains(key))
    {
        _keys.Add(key);
    }

    return result;
}
```

```
public void Remove(string key)
{
    var items = new List<string>() { key };
    if (key.Contains('*'))
    {
        items = _keys
            .Where(x => x.StartsWith(key.Replace("*", string.Empty)))
            .ToList();
    }

    foreach (var item in items)
    {
        cache.Remove(item);
        _keys.Remove(item);
    }
}
```

```

public class List
{
    public record ArticleListResponse(int TotalCount, IEnumerable<Response> Items) : AbstractListResponse<Response>(TotalCount, Items) { }

    public class Response
    {
        public Guid Id { get; set; }
        public string Slug { get; set; }
        public string Title { get; set; }
        public string Description { get; set; }
        public string Body { get; set; }
    }

    public class Endpoint : AbstractEndpoint
    {
        public override void MapEndpoint(IEndpointRouteBuilder app)
        {
            app
                .MapGet("/articles", Handle<ArticleListResponse, Query>)
                .WithName(nameof(List))
                .WithTags(nameof(Articles))
                ;
        }
    }

    public class Query : AbstractListQuery, ICacheable, IRequest<ArticleListResponse>
    {
        public string Slug { get; set; }
        public string Title { get; set; }
        public string Description { get; set; }

        public bool GetBypassCache() => false;
        public string GetCacheKey() => $"Articles::{GetBaseCacheKey()}";
        public TimeSpan? GetSlidingExpiration() => TimeSpan.FromHours(1);
    }
}

```



```
// No validator !
```

```
public class Handler(IRepositoryWithFind<Article> articleRepository) : IRequestHandler<Query, ArticleListResponse>
{
    public async Task<ArticleListResponse> Handle(Query request, CancellationToken cancellationToken)
    {
        var query = articleRepository.FindLike(
            new Dictionary<string, object>
            {
                { nameof(Article.Slug), request.Slug },
                { nameof(Article.Title), request.Title },
                { nameof(Article.Description), request.Description }
            }
        );

        List<Response> items = await request
            .ApplySkipTake(query)
            .Select(x => new Response
            {
                Id = x.Id,
                Slug = x.Slug,
                Title = x.Title,
                Description = x.Description,
                Body = x.Body
            })
            .ToListAsync(cancellationToken);

        return new ArticleListResponse(
            await query.CountAsync(cancellationToken),
            items
        );
    }
}
```

**< EUH... C'EST BIEN, MAIS
COMMENT JE METS TOUT ÇA
ENSEMBLE ? />**

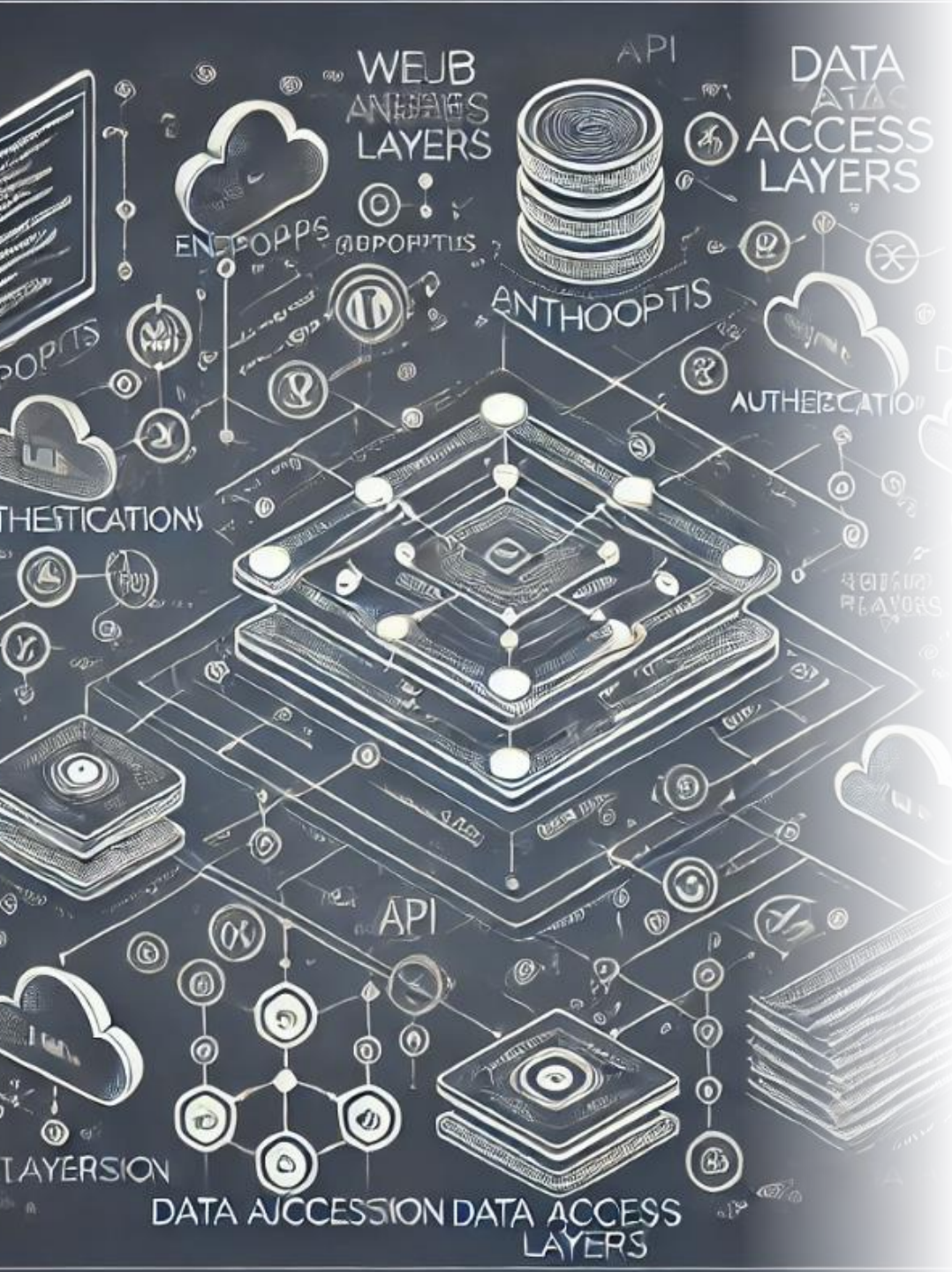
Architecture

+ Solution 'SimpleArchitecture'

Solution Items

- ▶ API
- ▶ Common
- ▶ Data
- ▶ ExternalServices
- ▶ Tests



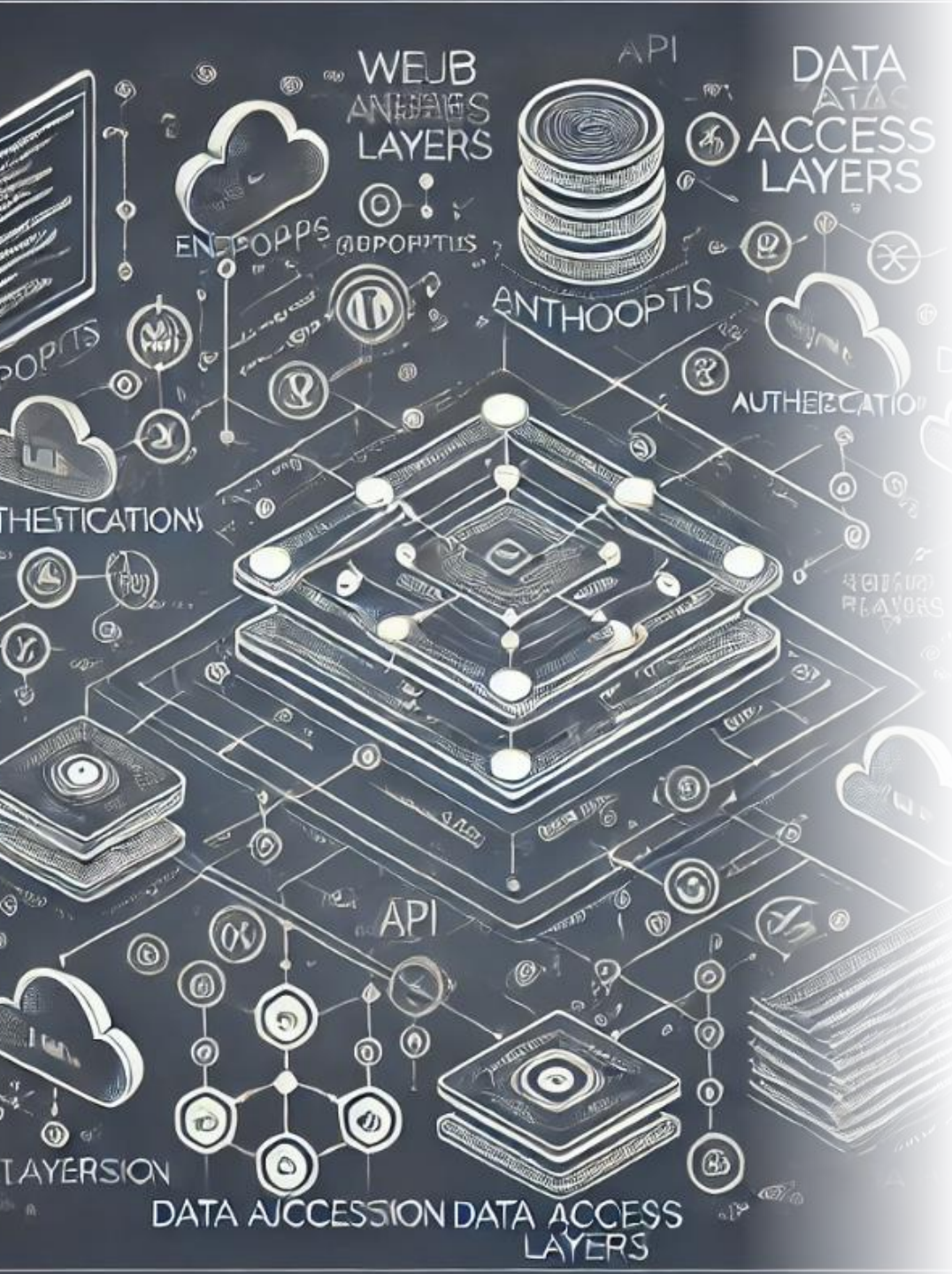


Architecture

- API
 - Connected Services
 - Dependencies
 - Properties
 - Behaviors
 - Features
 - Handlers
 - Middleware
 - Requests
 - Responses
 - Validators
 - .editorconfig
 - API.http
 - appsettings.json
 - ConfigureApp.cs
 - ConfigureServices.cs
 - Program.cs



Architecture



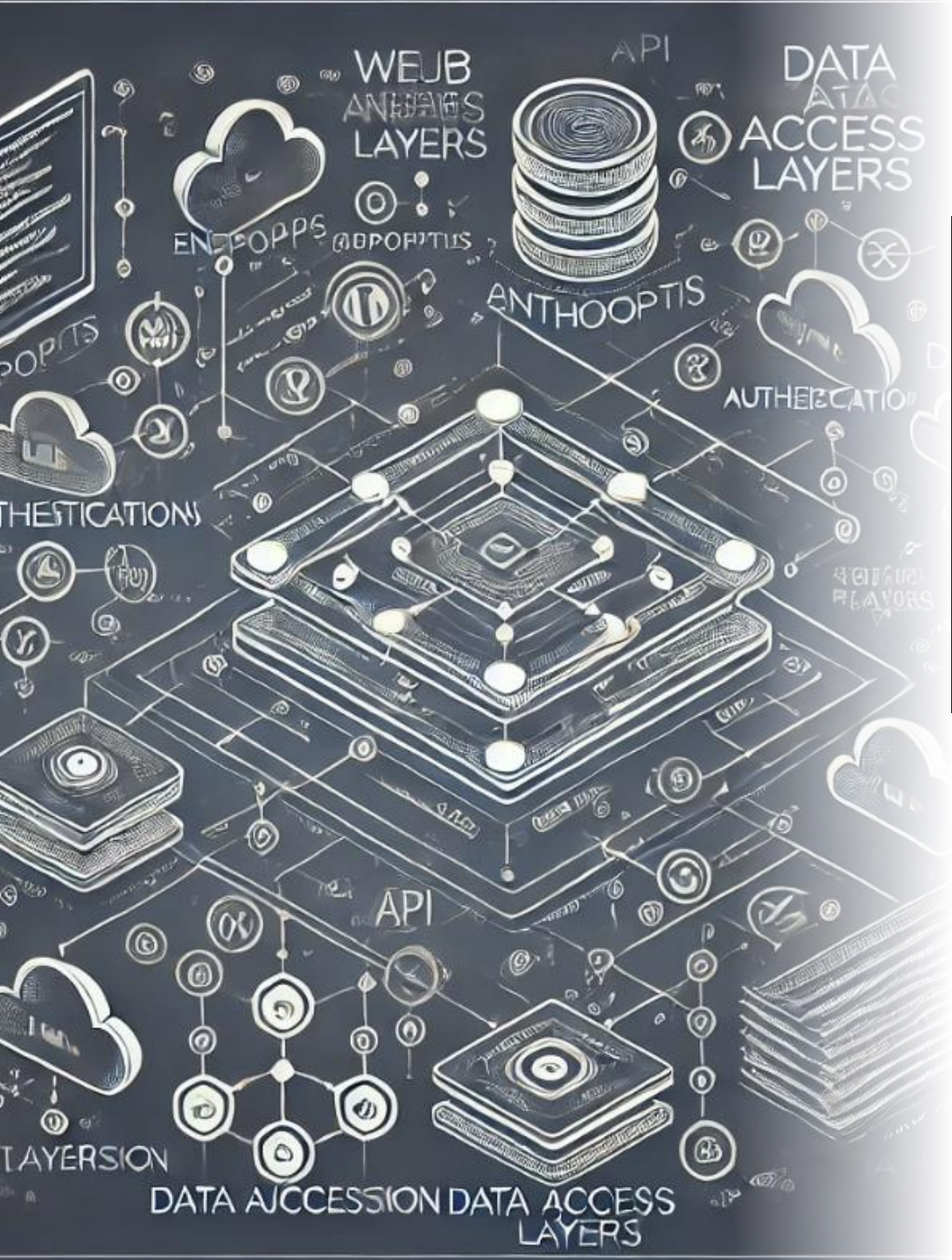
```

└─ [C#] Common
    ├── Dependencies
    ├── Exceptions
    ├── Extensions
    ├── Helpers
    ├── Providers
    ├── .editorconfig
    ├── C# Constants.cs
    └── C# ValidationErrors.cs

```



Architecture



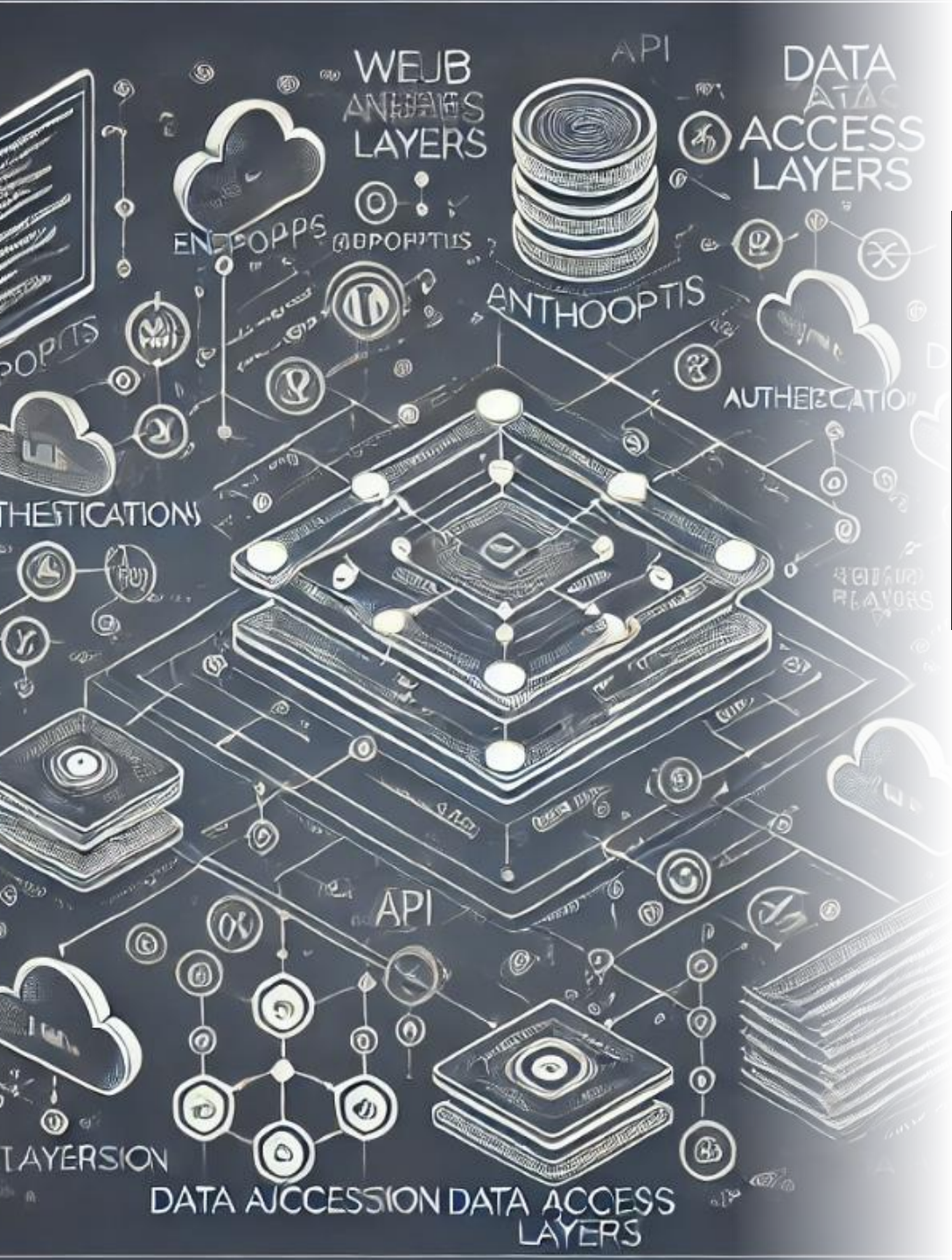
```

Data
├── Dependencies
├── Entities
├── Migrations
├── Repositories
├── SQL
├── .editorconfig
└── C# ApplicationDbContext.cs

```



Architecture



```
graph TD
    Tests --> Dependencies
    Tests --> IntegrationsTests[IntegrationsTests]
    IntegrationsTests --> Features
    Features --> Articles
    Articles --> CreateTestCs[+ C# CreateTest.cs]
    Tests --> UnitTests
```

- Tests
 - Dependencies
 - IntegrationsTests
 - Features
 - Articles
 - + C# CreateTest.cs
 - UnitTests



**<OUF ! ÇA FAIT BEAUCOUP À
COMPRENDRE !
EST-CE QU'ON PEUT AVOIR UN
RÉSUMÉ ? />**



- Séparation logique (CUD VS R)
- Command → Opération
- Query → Lecture

- Une librairie avec beaucoup de possibilités
- Pipelines/Behaviors selon vos besoins

- Basée sur l'expérience
- Sera toujours en constante évolution
- Répond pour les besoins d'aujourd'hui, mais...



DES QUESTIONS ?



N'oublier pas d'évaluer ma présentation !
Vos commentaires sont le bienvenu.



