CQRS in .NET

What is CQRS?

بص يا صديقى، CQRS ده اختصار لـ Command Query Responsibility Segregation، وده ببساطة نمط تصميمي (Design Pattern) بيقولك: "ليه تخلي القراءة والكتابة ماشيين في نفس السكة؟ افصلهم وريح دماغك!".

الفكرة إنك تخلي عمليات الكتابة (Commands) شغالة لوحدها، وعمليات القراءة (Queries) شغالة لوحدها، بحيث إن كل واحدة تبقى متخصصة في اللي بتعمله، وده بيخلي السيستم أسرع وأكتر كفاءة، خصوصًا لما العدد يكبر.

يعني مثلًا، بدل ما تيجي تبعت أمر للسيستم يقول "هات لي بيانات الطلب"، والسيستم يروح جايبها ويعدل فيها، لا! الـ Queries تقرأ بس، وبكده تضمن إن كل حاجة Commands والـ تكتب بس، وبكده تضمن إن كل حاجة منظمة ومافيش لخبطة بين اللي بيقرأ واللي بيعدل.

Why Use CQRS?

التوسّع (Scalability): تقدر توسّع عمليات القراءة والكتابة كل واحدة لوحدها من غير ما تأثر على التانية، وده بيخلي السيستم يشتغل بكفاءة حتى لو عدد المستخدمين زاد جدًا.

الأداء (Performance): عشان كل حاجة متقسمة، عمليات القراءة والكتابة بتشتغل بأفضل أداء من غير ما يعطلوا بعض، وده بيحسن سرعة الاستجابة في السيستم.

الأمان (Security): تقدر تتحكم في مين يكتب ومين يقرأ بشكل أوضح، وده بيدي أمان أعلى للبيانات، لأن الكتابة والقراءة متفصلين عن بعض.

المرونة (Flexibility): تقدر بسهولة تتعامل مع البيانات المعقدة وتغير طريقة إدارتها من غير ما تدخل فى دوامة التعديلات الكبيرة.

نموذج منفصل للكتابة والقراءة: كل واحدة ليها الـ Model الخاص بيها، يعني اللي بيكتب غير اللي بيقرأ، وده بيخلي الدنيا أنظف وأسهل في التطوير.

Basic Architecture of CQRS

بص يا صديقى، في CQRS الدنيا بتتقسم بين الكتابة والقراءة عشان السيستم يشتغل بكفاءة وما يحصلش دوشة بين الاتنين. تعالَ نشوف الحكاية ماشية إزاي:

1. Commands (الأوامر - الكتابة):

لما حد عايز يضيف حاجة، يعدّل، أو يمسح، بيبعت Command. الـ Command Handlers هما اللي بيتعاملوا مع الموضوع ده، وبيحفظوا البيانات في قاعدة بيانات الكتابة (Write Model DB).

1. Queries (الاستعلامات - القراءة):

لو حد عايز يشوف البيانات بس من غير ما يعدّل عليها، بيستخدم Query. الـ Query Handlers بيروحوا لقاعدة بيانات القراءة (Read Model DB) ويجيبوا المعلومات بسرعة.

1. نماذج بيانات منفصلة (كل واحد في حاله):

قاعدة بيانات الكتابة متصممة عشان تضمن إن كل حاجة ماشية صح ومتوافقة. قاعدة بيانات القراءة معمولة للسرعة، وممكن تكون حاجة زي NoSQL عشان تجيب البيانات في لمح البصر. كل واحدة مفصولة عن التانية عشان الأداء يبقى أحسن.

1. تدفّق البيانات في CQRS:

العميل (Client) يبعت Command لو عايز يغيّر حاجة. Command Handlers ياخدوا الطلب ويحفظوا البيانات في قاعدة الكتابة. بعد كده، البيانات دي تتحدّث في قاعدة القراءة، بحيث أي حد يطلب Query يلاقيها بسرعة. يعني الكتابة ماشية لوحدها والقراءة ماشية لوحدها، وكل واحد شغال في حتّته من غير ما يعطل التاني.

Implementing CQRS with Mediator Pattern

بص يا صديقى، في .NET لو عايز تطبق CQRS بطريقة مرتبة وسهلة، أشهر حاجة تستخدمها هي Mediator Pattern عن طريق MediatR Library، ودي بتساعدك تفصل بين الأوامر والاستعلامات من غير ما الدنيا تبقى متلخبطة. تعالَ نشوف الموضوع خطوة بخطوة.

```
// kamranrafiqkhan
public class CreateUserCommand : IRequest<User> {
    public string Name { get; set; }
}

public class GetUserQuery : IRequest<User> {
    public int Id { get; set; }
}

```

```
// kamranrafiqkhan
public class CreateUserHandler : IRequestHandler<CreateUserCommand, User> {
    public async Task<User> Handle(CreateUserCommand request, CancellationToken cancellationToken) {
        // Logic to create a user
    }
}

public class GetUserHandler : IRequestHandler<GetUserQuery, User> {
    public async Task<User> Handle(GetUserQuery request, CancellationToken cancellationToken) {
        // Logic to retrieve a user
    }
}
```

Registering MediatRin .NET

عشان تستخدم MediatR مع CQRS في مشروع .NET، لازم الأول تسجّله في ملف Startup.cs أو Program.cs حسب نوع التطبيق.

```
// kamranrafiqkhan
public void ConfigureServices(IServiceCollection services) {
    services.AddMediatR(typeof(Startup));
}
```

بص يا صديقي، بعد ما سجلنا MediatR في Program.cs أو Startup.cs، كده خلينا Dependency Injection (DI) تقدر تحقن الـ Handlers تلقائيًا من غير ما نربطهم يدويًا.

💡 يعني إيه الكلام ده؟

لما تستدعي IMediator.Send(command)، MediatR هيدور تلقائيًا على الـ Handler المناسب وينفذه، من غير ما تحتاج تكتب كود صريح لربط كل Command أو Query بالـ Handler بتاعه.

Using Commands and Queries

بص يا صديقى، عشان تنفّذ Command أو Query باستخدام MediatR، هتستخدم IMediator جوه الـ Controllers أو الـ Services، وده بيخلي الكود أنضف وأكتر تنظيمًا

```
// kamranrafiqkhan
public class UserController : ControllerBase {
    private readonly IMediator _mediator;

    public UserController(IMediator mediator) {
        _mediator = mediator;
    }

    [HttpPost]
    public async Task<IActionResult> CreateUser(CreateUserCommand command) {
        var user = await _mediator.Send(command);
        return Ok(user);
    }

    [HttpGet("{id}")]
    public async Task<IActionResult> GetUser(int id) {
        var user = await _mediator.Send(new GetUserQuery { Id = id });
        return Ok(user);
    }
}
```

بالظبط يا صديقى ! الـ CQRS مع MediatR بيخلّي الأوامر (Commands) والاستعلامات (Queries) منفصلين عن بعض، وده بيدي وضوح أكتر للكود وسهولة في الصيانة والتطوير. خلينا نوضح الفكرة دي بشكل عمل