**رویکرد CQRS ( Command Query Responsibility Segregation )**

**بررسی کلی رویکرد CQRS – Command Query Responsibility Segregation**

در یک سیستم نرم افزاری به صورت کلی درخواست های کلاینت به 4 گروه ( Create و Read و Update و Delete ) تقسیم می شوند و اصولا درخواست کلاینت از سرور و دیتابیس به یکی از این چهار حالت هست .

اگر فرض کنید منابع سیستم 100% باشد سهم هر کدام از 4 نوع درخواست 25% می باشد .

این تقسیم بندی در حالت عادی بهینه نیست . جهت بهینه سازی می تواند درخواست ها را به دو گروه تقسیم کرد و منابع سیستم را برای هر گروه تنظیم کرد .

گروه Command ( دستور ) که شامل Create و Update و Delete می باشد .

گروه Query ( پرس و جو ) که شامل درخواست های از نوع Read می باشد .

در حالت توسعه بهینه می توان سیستم را به گونه ای طراحی کرد که هرگروه یک دیتابیس مجزا داشته باشد و در پس زمینه دیتابیس ها همگام شوند ( Event ) .که تا زمانی که دیتابیس برای عملیات ثبت ( Create ) لاک ( lock ) می شود . درخواست های نوع Read معطل بازشدن لاک دیتابیس نشوند .

**اساس رویکرد CQRS جداکردن خواندن ها از نوشتن ها است .** که این می تواند در مقیاس یک کلاس باشد به گونه ای که در سطح کلاس خواندن از نوشتن جدا می شود و یا می توان در سطح فیزیکی باشد به گونه ای که خواندن ها در یک سرور و نوشتن ها در سرور دیگر **.**

**سوال - این رویکرد چه نوع مشکلی را حل می کند ؟**

به صورت کلی در طراحی دیتابیس عموما یا تمرکز روی نوشتن بهینه هست و یا روی خواندن بهینه . در اغلب موارد یکی فدای دیگری می شود . اما برای یک توسعه همه جانبه باید هم به خواندن بهینه و هم به نوشتن بهینه توجه شود .

در اینجا رویکرد CQRS به ما اجازه می دهد تمرکز جداگانه ای روی نوشتن و خواندن داشته باشیم . روی خواندن تمرکز کنیم بدون توجه و نگرانی درباره نوشتن و همچنین روی نوشتن تمرکز کنیم بدون نگرانی روی خواندن . به عبارت دیگر در یک سیستم پیاده سازی شده با CQRS در سطح بالا شما ممکن هست بر حسب نیاز طراحی دیتابیس برای نوشتن از نوع Relational و طراحی دیتابیس برای خواندن از نوع document استفاده کنید .

در این حالت توسعه نرم افزار در مقیاس های بزرگ بسیار ساده باشد .

مشخصا این رویکرد مشکلات خود را دارد . از جمله مدیریت سیستم های جدا ، دیتاهای قدیمی ، سینک کردن دیتابیس ها و ... اما حل همه این چالش ها زمانی منطقی می شود که ما در سیستم نیاز به بهینه سازی داشته باشیم و در صورت نیاز به آن رجوع کنیم بهتر است همیشه در فکر توسعه آسان باشیم و تنها در زمان های ضروری به فکر توسعه پیچیده و استفاده از رویکردی مثل CQRS باشیم .

**الگوی Mediator**

اساس این الگو که در کنار رویکرد CQRS کارآمد هست ، جداسازی وابستگی بین اشیاء هست .

برای مثال در الگوی CQRS ما درخواست داریم و یک پردازش گر درخواست .

* Request Query و Request Query Hanler
* Request Command و Request Command Handler

به صورت کلی نیاز نیست که شی فرستنده درخواست ( Request Model ) چیزی درباره پردازشگر درخواست ( Request Handler ) بداند . این جداسازی با استفاده از الگوی Mediator امکان پذیر هست .

در دات نت کتابخانه MediatR برای این منظور استفاده می شود .

Install-Package MediatR

منابع :

<https://code-maze.com/cqrs-mediatr-in-aspnet-core/>