**CQRS**

CQRS is an acronym for Command and Query Responsibility Segregation. The central concept of this pattern is that an application has read operations and write operations that must be totally separated. This also means that the model used for write operations (commands) will differ from the read models (queries). Furthermore, the data will be stored in different locations. In a relational database, this means there will be tables for the command model and tables for the read model. Some implementations even store the different models in totally different databases, e.g. SQL Server for the command model and MongoDB for the read model.

CQRS

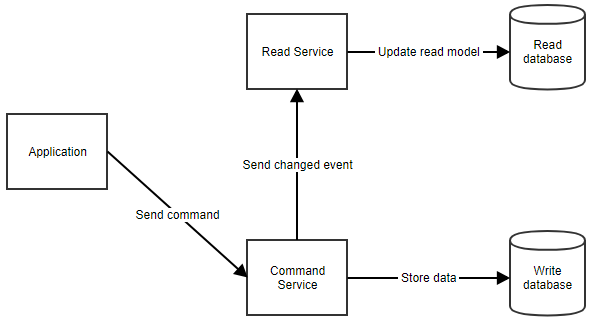
CQRS مخففی برای تفکیک فرماندهی و مسئولیت پرس و جو است. مفهوم اصلی این الگوی این است که یک برنامه دارای عملیات خواندن و نوشتن است که باید کاملاً از هم جدا شوند. این همچنین بدان معنی است که مدل مورد استفاده برای عملیات نوشتن (دستورات) با مدلهای خوانده شده (نمایش داده شد) متفاوت خواهد بود. علاوه بر این ، داده ها در مکان های مختلف ذخیره می شوند. در یک پایگاه داده رابطه ای ، این بدان معناست که جدول هایی برای مدل فرمان و جداول برای مدل خواندن وجود خواهد داشت. برخی از پیاده سازیها حتی مدلهای مختلف را در پایگاههای داده کاملاً متفاوت ذخیره می کنند ، به عنوان مثال. SQL Server برای مدل فرمان و MongoDB برای مدل خواندن.

This pattern is often combined with event sourcing, which we’ll cover below.

How does it work exactly? When a user performs an action, the application sends a command to the command service. The command service retrieves any data it needs from the command database, makes the necessary manipulations and stores that back in the database. It then notifies the read service so that the read model can be updated. This flow can be seen below.

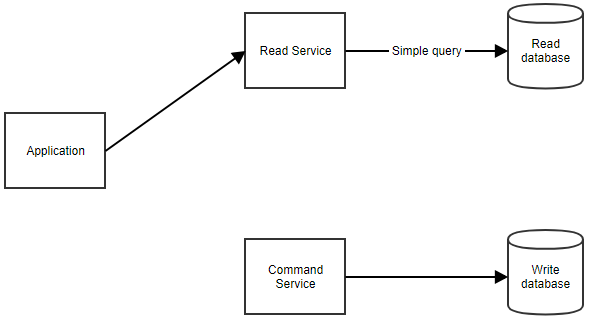
این الگوی غالباً با استفاده از event sourcing ترکیب می شود.

دقیقا چطوری کار می کند؟ هنگامی که کاربر عملی را انجام می دهد ، برنامه یک دستور به سرویس فرمان ارسال می کند. سرویس فرمان هرگونه اطلاعات مورد نیاز خود را از پایگاه داده فرمان بازیابی می کند ، دستکاری های لازم را انجام داده و بلاک را در پایگاه داده ذخیره می کند. سپس سرویس خواندن را آگاه می کند تا بتواند مدل خواندن را به روز کند. این جریان را می توان در زیر مشاهده کرد.



When the application needs to show data to the user, it can retrieve the read model by calling the read service, as shown below.

هنگامی که برنامه باید داده ها را به کاربر نشان دهد ، می تواند با تماس با سرویس خواندن همانطور که در زیر آمده است ، مدل خواندن را بازیابی کند.

****

**Advantages**

* Command models can focus on business logic and validation while read models can be tailored to specific scenarios.
* You can avoid complex queries (e.g. joins in SQL) which makes the reads more performant.

مزایا

• مدلهای فرمان می توانند بر منطق كسب و كار و اعتبار سنجی تمرکز كنند در حالی كه مدلهای خوانده شده می توانند متناسب با سناریوهای خاص باشند.

• می توانید از پرس و جوهای پیچیده (به عنوان مثال پیوستن به SQL ) جلوگیری کنید که باعث می شود عملکرد خواننده بیشتر شود.

**Disadvantages**

* Keeping the command and the read models in sync can become complex.

معایب

• نگه داشتن دستور و مدلهای خوانده شده در همگام سازی می تواند پیچیده باشد.

**Ideal for**

* Applications that expect a high amount of reads
* Applications with complex domains

مناسب برای

• برنامه هایی که انتظار زیاد خواندن را دارند

• برنامه های کاربردی با دامنه های پیچیده

<https://dzone.com/articles/software-architecture-the-5-patterns-you-need-to-k>