1. **CQRS (Common and Query Responsibility Segregation)**

In CQRS, we split the application into two parts: **Command-Side** and **Query-Side.**

* **Commands** change the state of the object or entity, also called modifiers or mutators.
* **Queries** return the state of the entity and do not change anything. Another term for them is “accessors”.

**Why Is it required?**

In traditional data management systems, **both commands and queries are executed against the same set of entities**, having a single representation or view. Different clients want different views across sets of data. CRUD operations are applied to a single datastore and the **same entity or object is accessed to handle both read and write operations.**

There are issues with having a single view for both read and write sides.

* Introduces the risk of data contention.
* Managing permissions and security become complex as the same objects are exposed to both read and write operations. Different clients that are updating the data unknowingly conflict.

**How CQRS sloves This Problem ??**

The CQRS pattern holds the idea that ***the method should either change the state of the entity or return the result, but not both***. Segregating models for the read and write sides reduces the complexity that comes with having a single view for both of them.

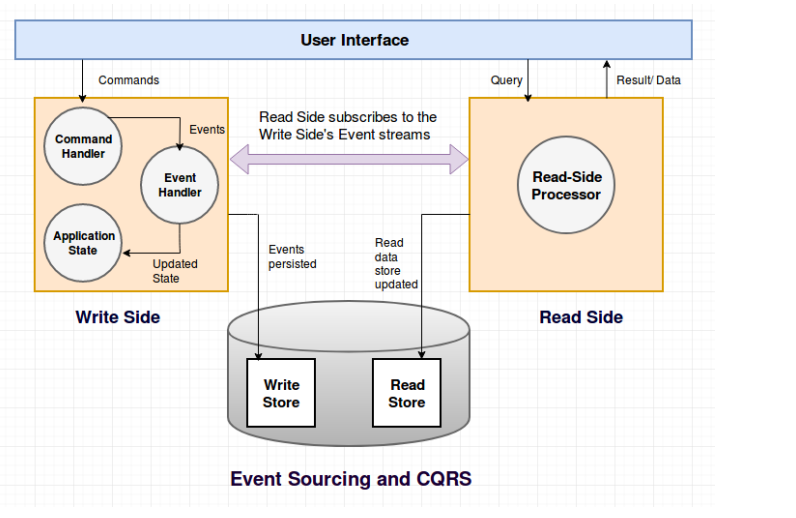
**Benefits of CQRS:**

* **Separate command and query models**, resulting in simplified design and implementation of the system and overall reduction of complexity.
* One can easily optimize the read side of the system separately from the write side, allowing scaling of each differently as per the load on the side. For example, read datastores often encounter greater load, and hence can be scaled without affecting the write datastores.
* Can provide multiple views of your data for querying purposes depending on the use cases.

**How CQRS Works**

CQRS is ***mainly used in conjunction with Event Sourcing***. The **write side** model of the CQRS-based system **handles the events persistence**, **acting as a source of information for the read side.** The read model of the system provides materialized views of the data, typically as hightly denormalized views.

Details of CQRS-based system :



Refernce Site: <https://www.kindsonthegenius.com/microservices/understanding-the-files-and-components-of-a-cqrs-applicationwith-axonframework/>