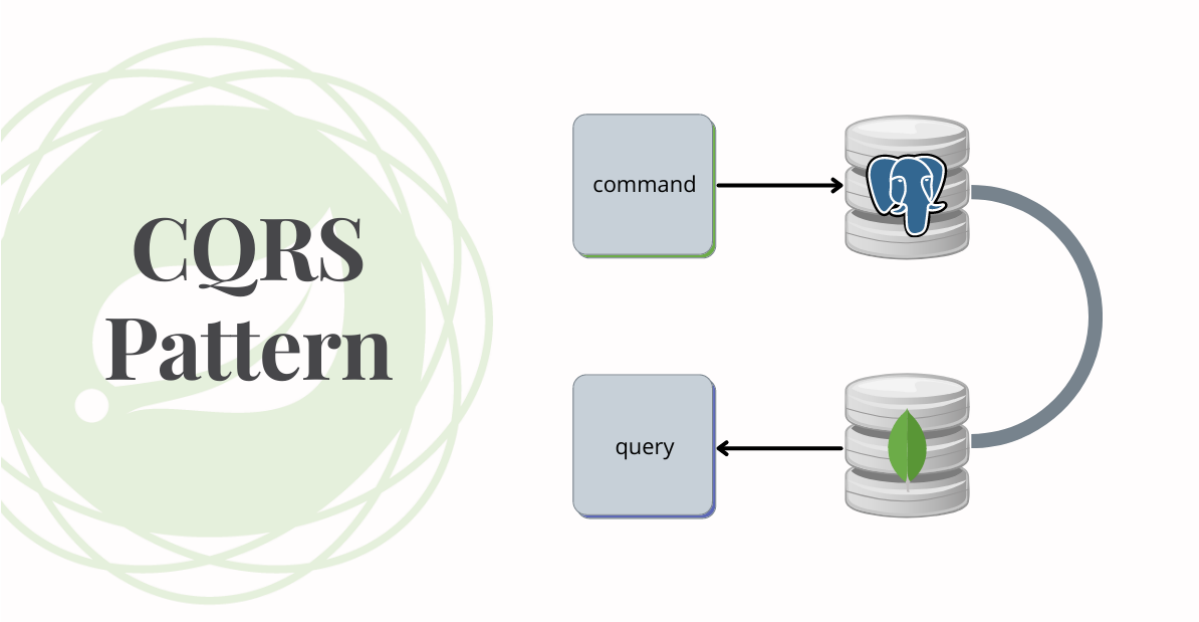
Universidad de la Cuenca del Plata

Ingeniería en Sistemas de Información

CQRS

Paradigmas y Lenguajes de Programación III



Profesor: José Fernandez

Integrantes:

- Dikowiec Sebastian
- Ferro Santiago
- Godoy Juan
- Zalazar Matías

Año: 2022

¿Que es un patrón?

En informática, un patrón es una técnica que permite la resolución de problemas de distinta índole, como por ejemplo, problemas de comportamiento, problemas estructurales, o bien, problemas creacionales.
En este caso estamos tratando sobre un patrón que ayuda a solucionar problemas en relación a sistemas distribuidos, los cuales son conjuntos de programas informáticos que utilizan recursos computacionales en varios nodos de cálculo distintos para lograr un objetivo compartido común y tienen la finalidad de eliminar los puntos de error centrales de un sistema.

¿Qué es CQRS?

Las siglas CQRS vienen del inglés y significan “Command query responsibility segregation”, en español, la separación en la responsabilidad de las lecturas y los comandos.
Es un patrón que separa las operaciones de lectura y actualización de un almacén de datos. La implementación de CQRS en la aplicación puede maximizar el rendimiento, la escalabilidad y la seguridad. La flexibilidad creada al migrar a CQRS permite que un sistema evolucione mejor con el tiempo y evita que los comandos de actualización provoquen conflictos de combinación en el nivel de dominio.

Implementación

Estos son algunos de los desafíos de la implementación de este patrón:

- **Complejidad:** La idea básica de CQRS es sencilla, pero puede generar un diseño de aplicación más complejo, especialmente si incluye el patrón Event Sourcing.El uso de CQRS viene muy ligado al uso de “Event Sourcing” en la base de datos de escritura.
- **Sincronizar:** la base de datos de escritura con la de lectura para que no haya ninguna información diferente. Por ello deberemos hacer uso de un service bus.
- **Mensajería:** Aunque CQRS no requiere mensajería, es normal usarla para procesar comandos y publicar eventos de actualización. En ese caso, la aplicación debe gestionar errores de mensaje o mensajes duplicados.
- **Coherencia final:** Si separa las bases de datos de lectura y escritura, los datos de lectura pueden estar obsoletos. El almacén de modelos de lectura debe actualizarse para reflejar los cambios del almacén de modelos de escritura, y puede ser difícil detectar cuándo un usuario ha emitido una solicitud basada en datos de lectura obsoletos.

¿Cuándo usarlo?

Es óptimo implementar CQRS en estos escenarios:

- Dominios colaborativos en los que muchos usuarios acceden a los mismos datos en paralelo. CQRS permite definir comandos con la suficiente granularidad para minimizar los conflictos de combinación en el nivel de dominio (cualquier conflicto que surja se podrá combinar mediante el comando).
- Interfaces de usuario basadas en tareas en las que se guía al usuario a través de un complejo proceso como una serie de pasos o con modelos de dominio complejos. El modelo de escritura tiene una pila de procesamiento de comandos completa con lógica de negocios, validación de entrada y validación empresarial. El modelo de escritura puede tratar un conjunto de objetos asociados como una única unidad para los cambios de datos (un agregado, en la terminología de DDD) y asegurarse de que

estos objetos siempre están en un estado coherente. El modelo de lectura no tiene ninguna lógica de negocios ni pila de validación y solo devuelve un DTO para su uso en un modelo de vista. El modelo de lectura tiene coherencia final con el modelo de escritura.

- Escenarios en los que el rendimiento de las lecturas de datos se debe ajustar por separado del rendimiento de las escrituras de datos, especialmente cuando el número de lecturas es mucho mayor que el número de escrituras. En este escenario, puede escalar horizontalmente el modelo de lectura, pero ejecutar el modelo de escritura solo en algunas instancias. Un número reducido de instancias de modelo de escritura también ayuda a minimizar la aparición de conflictos de combinación.
- Escenarios en los que un equipo de desarrolladores pueda centrarse en el modelo de dominio complejo que forma parte del modelo de escritura, y otro equipo pueda centrarse en el modelo de lectura y en las interfaces de usuario.
- Escenarios en los que se espera que el sistema evolucione con el tiempo y que podrían contener varias versiones del modelo, o en los que las reglas de negocio cambian con regularidad.
- Integración con otros sistemas, especialmente en combinación con Event Sourcing, en los que el error temporal de un subsistema no debería afectar a la disponibilidad de los demás.

No se recomienda este patrón si:

- El dominio o las reglas de negocio son simples.
- Es suficiente con una interfaz de usuario simple, de estilo CRUD, y las operaciones relacionadas de acceso a datos.

Ventajas

- **Ambos subsistemas pueden evolucionar por separado:** el mantenimiento y despliegue puede estar diferenciado. Esto es una ventaja porque normalmente el ritmo de cambios en la funcionalidad es muy diferente en las consultas y los comandos. En mi experiencia es más frecuente añadir una nueva consulta que un nuevo comando.
- **Ambos sistemas pueden escalar por separado:** Si nuestro negocio es muy intensivo en lecturas, podemos dedicar más máquinas al subsistema de consultas de forma sencilla. Simplemente levanta otra instancia y conéctala de forma que pueda recibir los cambios transmitidos por el subsistema de comandos.
- **El estilo CQRS es modular de forma inherente:** Podemos escoger ir a una arquitectura modular, donde haya varios subsistemas de consulta y varios subsistemas de comandos. Cada uno de ellos de nuevo puede ser diseñado, mantenido, escalado y desplegado por separado. Es muy interesante la posibilidad de añadir nueva funcionalidad (nuevos subsistemas de comandos o consultas) sin tener que parar el resto del sistema.

Implementaciones en C

1 - https://learn.microsoft.com/es-es/azure/architecture/patterns/cqrs
2 - https://www.netmentor.es/entrada/patron-cqrs-explicado-10-minutos

Presentacion

QCRS

https://docs.google.com/presentation/d/1ifF8WkFhd4Jay7rz2MX1OZ_dRtggsWVB_wAszcZQeh4/edit#slide=id.p