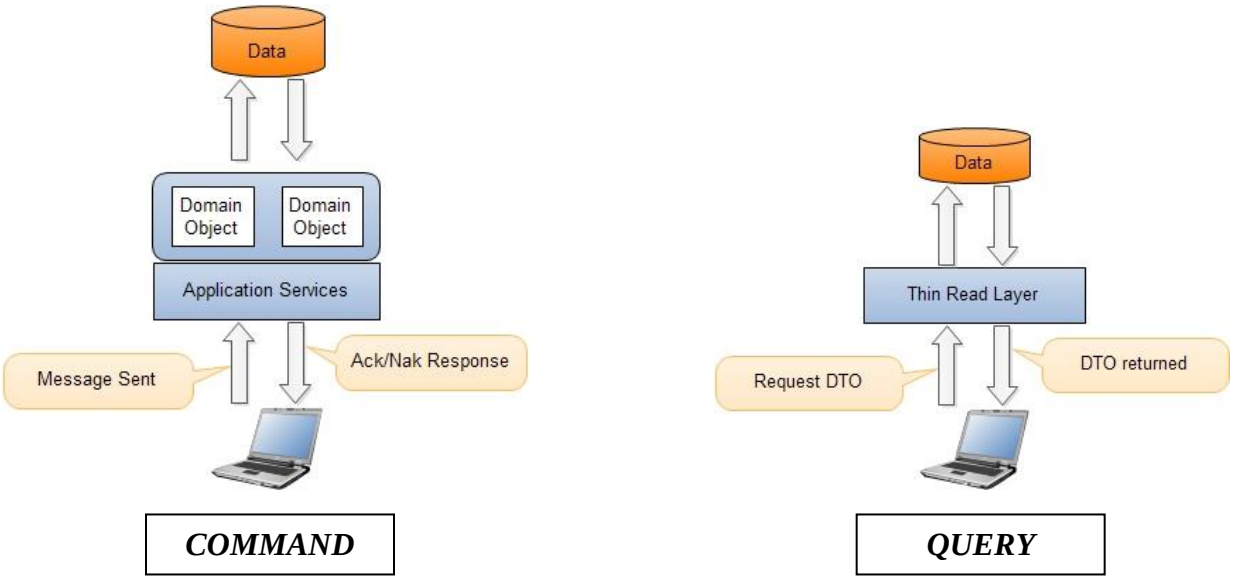
CRĂCIUNESCU Ligia – Izabela
ANUL IV, IS, Semigrupa 2.1

Command Query Responsibility Segregation

Command Query Responsibility Segregation *(CQRS)* este un șablon arhitectural care a fost discutat prima dată de ***Greg Young*** în anul 2010. Acesta folosește ca bază principiul enunțat de către ***Bertrand Meyer***, care a afirmat că orice metodă trebuie să fie ori o **Interogare** (doar citește informații) ori o **Comandă** (execută o acțiune), dar nu ambele simultan și îl completează, sugerând ca responsabilitățile de citire și cele de scriere să fie împărțite în obiecte complet distincte. Această abordare se opune celei tradiționale **CRUD**, unde ***Modelul*** este creat, citit, modificat și șters în mod uniform în cadrul aceleiași clase, cu metode pentru fiecare dintre aceste acțiuni.

Prin urmare, şablonul spune că nu este necesară aceeaşi sursă de stocare de date şi nici măcar aceeaşi tehnologie pentru mecanismele de scriere şi citire. Acest lucru permite așadar scalarea lor independentă. Cele mai multe dintre aplicații necesită citiri intensive, astfel că serviciile dedicate de interogare pot fi scalate orizontal folosing zeci de mașini. În același timp, serviciile de comenzi trebuie scalate la o scară mult mai mică, așadar vor fi necesare mai puține mașini. CQRS se potrivește foarte bine cu modelele bazate pe evenimente, deoarece permite ***event sourcing***-ului să înlocuiască baza de date consistentă cu un ***storage*** pentru evenimente. Evenimentele sunt apoi aplicate și se obțin datele interogabile. Un avantaj mai puțin intuitiv vine din ușurința cu care se pot împărți ***task***-uri atomice programatorilor din echipă. Astfel, echipele mai numeroase pot colabora mai bine pe cod. În timp ce componentele CRUD pot fi încredințate colegilor mai puțin experimentați, complexitatea CQRS poate fi manevrată de colegii seniori, care apoi distribuie informația întregii echipe.

Şablonul recunoaşte că interogarea şi procesarea comenzilor sunt fundamental diferite:



- Procesarea comenzilor respectă reguli de business (engl. *business rules*) uneori extrem de complexe care dictează setul de combinaţii valide pentru manipularea datelor. Pentru procesarea comenzilor, considerentele de consistenţă şi integritate sunt esenţiale.
- Presupunând folosirea unei baze de date SQL pentru stocare, normalizarea este foarte importantă pentru procesarea comenzilor.
- Procesarea comenzilor depinde de consistenţa tranzacţională a datelor.
- Procesarea comenzilor poate fi asincronă , ceea ce reprezintă câteodată un beneficiu important de obţinut.
- Interogările nu au cerinţe de integritate, deci nu este nevoie nici de normalizare. Bazele de date denormalizate sunt o alegere excelentă, pentru că au interogările mai rapide.
- Interogările necesită filtre complexe şi agregări de date în beneficiul interfeţei grafice (engl. *GUI - Graphical User Interface*).

Totuși, există anumite ***riscuri*** în aplicarea acestui șablon, precum faptul că posibilitatea de a avea o bază de date pentru citire diferită de baza de date pentru scriere, cât și existența a două modele, unul pentru interogare şi unul pentru procesarea comenzilor, implică timp de acomodare până când va putea fi aplicată corespunzător. Totodată, complexitatea aplicației se accentuează pentru că datele modificate nu sunt imediat disponibile pentru prezentare odată ce comanda s-a executat.

Din punct de vedere al ***securității***, aceasta nu este responsabilitatea directă a niciunuia dintre nivele - de interogare sau de procesare a comenzilor. Dacă vorbim despre informația afișată, securitatea poate fi de regulă implementată ca o serie de filtre impuse. Decizia de a impune filtre pe date nu e facută de nivelul de interogare ci mai degrabă de un nivel superior. Cu privire la securitatea în ceea ce priveşte manipularea datelor de către un actor, aceasta pare mai degrabă o problemă de scenarii de utilizare (engl. use cases), deoarece fiecare actor va fi încadrat în unul din rolurile predefinite ale aplicaţiei. Prin urmare, nivelul de comandă nu e responsabil cu asigurarea securităţii ci doar cu execuţia comenzilor. În concluzie, decizia de a utiliza CQRS nu are un impact foarte mare asupra considerentelor de securitate.

Beneficiul maxim adus de CQRS apare atunci când aplicaţia este complexă şi ar avea nevoie de ***scalare*** dar, oricum, CQRS are avantaje chiar şi în cazurile care nu implică scalabilitate deoarece pe lângă aceasta, șablonul permite paralelizarea efortului de dezvoltare pe două nivele, permite menţinerea unui ***domain model*** corect şi pur pe partea de procesare a comenzilor, care să nu fie afectat de cerinţele de interogare (acestea s-ar putea schimba relativ frecvent atât în timpul cât şi după implementarea aplicaţiei), permite, de asemenea, folosirea tehnologiilor adecvate pentru procesarea comenzilor şi interogărilor, ***reducând astfel complexitatea aplicaţiei şi durata de dezvoltare şi îmbunătăţindu-i performanţa***, reduce complexitatea aplicaţiei prin separarea responsabilităţilor la nivel macro, permite folosirea diferitelor mecanisme de persistenţă precum ***event sourcing***, care este foarte potrivit în anumite cazuri, etc.