**Command Query Responsibility Segregation**

*Command Query Responsibility Segregation (CQRS)* reprezintă un model arhitectural care separă citirea și scrierea în doua modele disticte. Această abordare duce la faptul că fiecare metodă ar trebui să fie fie o comandă care are rolul de a efectua o acțiune, fie o interogare care returnează date. De reținut este că o comandă nu poate returna date, iar o interogare nu poate modifica datele.

CQRS are ca principiu de bază folosirea a două modele diferite de citire si scriere. Cu toate acestea, pentru a implementa aceast principiu putem acționa în mai multe moduri.

Pe de o parte, se poate utiliza o singura baze de date la care să fie conectate cele două modele. Atunci când un utilizator va efectua o comandă, modelul de scriere va actualiza baza de date. Iar apoi modelul de citit va interoga baza de date și va prezenta informațiile Interfeței Utilizator. Această abordare este una recomandată pentru aplicații care necesită procesare sincronă și rezultate imediate.

Pe de o altă parte, se pot folosi două stocări diferite de date. În acest caz, modelul de scriere va trimite comenzi la o singuă bază de date, iar apoi se va face o actualizare asincrona cu baza de date de citit. Această situație are beneficii ca:

* Stocarea datelor în baza de citire ca date denormalizate. Acest lucru duce la avantajul de a scrie mai multe interogări eficiente și de a stoca date care au mai mult sens în ceea ce privește modul de afișare în aplicație.
* Scalarea separată a două părți disticte ale apicației. Dacă va aflați în situația în care numărul de citiri este mult mai mare decât numărul de scrieri, puteți scala partea de interogare a aplicației fără a ține cont prea mult de partea de scriere. Se folosește atunci când sunt necesare interogări intense de raportare.

Pentru a implementa Command Query Responsibility Segregation se folosesc multe dintre aceleași modele. În continuare am să prezint câteva aspecte comune ale implementării:

* *Interfața utilizator bazată pe sarcini*. În aplicațiile des întâlnite CRUD, interfața cu utilizatorul nu este foarte dezvoltată și trebuie să ajute utilizatorul să atingă un obiectiv.

Astfel se pierde intenția utilizatorului. Deoarece, CQRS nu este axată pe CRUD, se permite scrierea unei interfeșe UI bazată pe sarcini. Rolul acesteia este de a reduce întreaga aplicație pentru a pune la dispoziția utilizatorului o interfață bazată pe intenții. Asta înseamnă că, în locul interfeței care se conformează arhitecturii aplicației, va fi dezvoltată o interfața de utilizator CQRS care va fi creeată în așa fel încât să rezolve nevoile utilizatorului.

* *Prelucrarea comezilor*. Atunci când folosim CQRS, modelul de scriere și modelul de citit sunt separate, iar cererile pentru procesare necesită un model mental ușor diferit. Spre exemplu, un model des întâlnit pentru afișarea unei resurse noi create este redirecționarea. Însă, atunci când modelul de citire este diferit de cel de scriere, pentru resursa nou creată nu se returnează id. În acest caz, se folosește un Bus Command pentru a face față fluxului de informații. Acest Bus Command nu trebuie să returneze nimic, deoarece în caz contrar

1

ar distruge separarea dintre modelul de citire si cel de scriere. Cu alte cuvinte este interzis ca nivelul de interogare să facă modificări în date.

* *Sincron sau asincron*. Procedura sincronă returneză imediat un raspuns utilizatorului și este folosită atunci când interfața cu utilizatorul necesită un rezultat instantaneu. Procedura asincrona nu generează un raspuns imediat și este utilă atunci când datele trebuie să treacă printr-o perioadă lungă și complexă de procesare. CQRS permite scrierea datelor într-o singură baze de date, procesarea lor asincrona și apoi trasnferul către baza de date citită. Cea din urmă procedură este cunoscută ca Eventual Consistency.
* *Evenimente de domeniu.* Pentru aplicațiile care conțin CQRS, evenimentele de domeniu sunt foarte importante, deoarece ele ascultă anumite evenimente în timpul execuției aplicației și oferă posibilitatea de a decupla consecințele unei actiuni în sine.

Pe lângă avantajul de a jongla între accesul de scriere și cel de citire, CQRS are numeroase beneficii:

* Permite nivelelor de scriere și citire să evolueze diferit în timp
* Datorită acestei separări, este posibil ca programatorii seniori să lucreze la domain modelul pentru procesarea comenzilor, în timp ce juniorii lucrează la achiziţia de date pentru GUI (nivelul de interogare).
* Partea de procesare a comenzilor este simplă și specialzată în respectarea regulilor de business, fără a fi influenșată de GUI
* Partea de interogare este simplă şi specializată în filtrarea şi agregarea datelor, în beneficiul GUI-ului, fără restricţii referitoare la modelul folosit pentru scriere.
* Permite ca dezvoltarea GUI-ului şi a părţii de interogare să fie făcute în paralel cu dezvoltarea părţii de procesare a comenzilor.
* În anumite circumstanţe potenţează aplicarea şablonului domain-driven design (care este în sine un aspect foarte important).
* Permite folosirea diferitelor mecanisme de persistenţă precum event sourcing , care este foarte potrivit în anumite cazuri.

Pe langă beneficii, CQRS prezintă și riscuri, ca:

* Ca orice șablon nou, prezintă schimbări ce implică timp pentru acomodare până va putea fi dezvoltat corescupunzător: baza de date pentru citire poate fi diferită de baza de date pentru scriere; existența a doua modele distincte (unul pentru interogare și unul pentru scriere)
* Complexitatea aplicației se accentuaza, fiindcă datele modificate nu sunt disponibiule imediat pentru prezentare, odată ce comanda a fost executată.

În concluzie, utilizarea CQRS adaugă aplicației web o complexitate ridicată și nu este indicat ca oricine să încerce să aplice aceast șablon. Această metodă are avantaje multiple și se recomandă să fie folosită pentru rezolvarea problemelor dificile și pentru aplicații complexe, dezvoltate la scară largă.

2