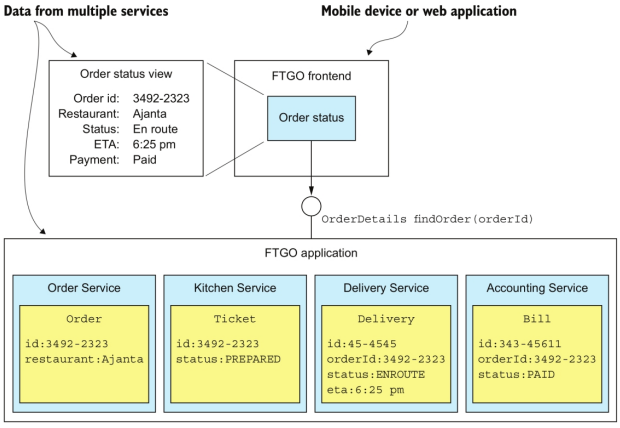
API composition, CQRS

Servisno orijentisane arhitekture

Univerzitet u Novom Sadu
Fakultet tehničkih nauka

Problem

- Formiranje upita u monolitnoj aplikaciji je često jednostavno zbog toga što ona poseduje jednu bazu podataka - svi neophodni podaci nalaze se na istom mestu
- PIsanje upita u mikroservisnoj aplikaciji je izazovniji zadatak zbog toga što se podaci često nalaze u različitim bazama podataka koje poseduju različiti servisi
- Veliki broj upita ne možemo rešiti jednim upitom ka bazi, servisi moraju međusobno da komuniciraju

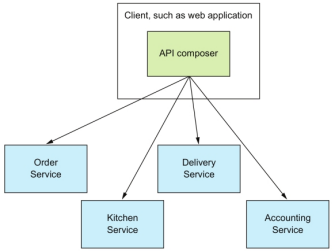


API composition

- API composition je jedan od šablona koji rešava navedeni problem
- Upit se formira tako da se iz jedne tačke iniciraju svi zahtevi koji dobavljaju neophodne podatke, a da se zatim na istom tom mestu rezultati agregiraju u jednu celinu
- Komponenta koja obavlja navedeni proces naziva se API composer, dok se servisi iz kojih se dobavljaju podaci nazivaju provajder servisi

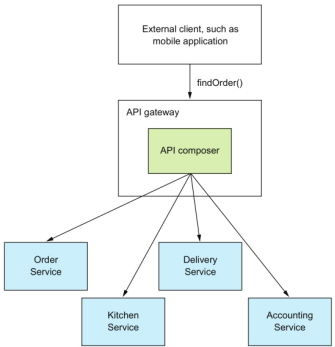
API composer - Klijent

- Prva opcija je da se neophodna logika ugradi u klijenta aplikacije
- Ovakvo rešenje zahteva od klijenta da kontaktira svaki servis, a zatim agregira dobijene rezultate



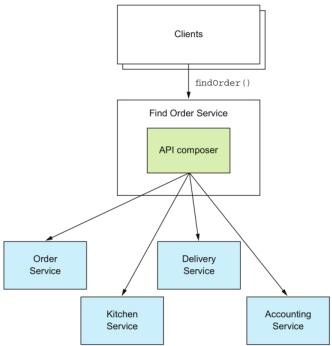
API composer - API gateway

- Druga opcija je da API Gateway bude zadužen za obavljanje kompozicije
- Ovakva organizacija ima smisla u situaciji kada je rezultat API Composition-a deo eksternog API-ja sistema i koriste ga klijenti
- Benefit je da se pristigli zahtev ne mora dalje rutirati nekom drugom servisu za agregaciju, a klijent jednim upućenim zahtevom dobija agregiran rezultat



API composer - Dedicated servis

- Treća mogućnost je da se API composer razvije kao novi servis kom je to primarna uloga
- Ovu strategiju pogodno je koristiti kada se rezultat upita koristi interno među ostalim servisima
- Kao i kada je logika agregiranja previše kompleksna da bi bila deo API gateway-a



Nedostaci

- Iako je vrlo jednostavan i često primenljiv šablon, nije u svim situacijama pogodan za upotrebu
- U nekim situacijama spajanje podataka u memoriji je veoma skupo ili nemoguće zbog veličine tih podataka, kada je bolje primeniti CQRS šablon
- Dodatno trošenje računarskih i mrežnih resursa jer je potrebno izvršiti više zahteva i više upita ka bazi
- Rizik od smanjene dostupnost zbog toga što barem tri komponente učestvuju u čitavom procesu
- Nekonzistentnost u podacima izazvana time što različiti servisi mogu posedovati različite verzije podataka u određenom trenutku

Primer

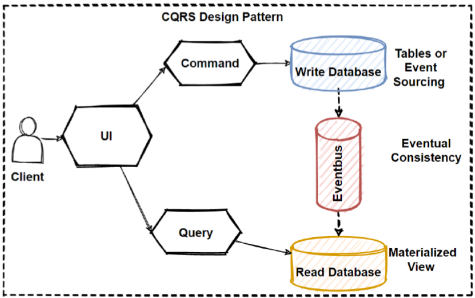
- Aplikacija podržava rad sa porudžbinama, prva asocijacia — klasičan web shop
- *ordering-service* omogućava dobavljanje porudžbina po identifikatoru, ali samo onih atributa porudžbine koje taj servis čuva
- Pošto hoćemo da istovremeno prikažemo i informacije o stanju i adresi isporuke, koristimo API composition šablon da dobavimo potrebne podatke iz *shipping* servisa
- API composer se nalazi u *ordering* servisu

Problem

- API composition šablon postaje izuzetno neefikasan kada treba da dobavimo i spojimo veliku količinu podataka u memoriji (na primer kreiranje čitave istorije porudžbina za nekog ili sve korisnike)
- I u situaciji kada se svi neophodni podaci za formiranje upita nalaze u samo jednom servisu, način na koji se oni skladište ne mora biti odgovarajući za taj upit (jedan model podataka nam nije dovoljan)
- Na primer podatke o tvitovima čuvamo u wide-column bazi, a želimo da formiramo upit za preporuku drugih tvitova na osnovu sličnosti novog sadržaja i onog na koji je korisnik pozitivno regovao u prošlosti (možda graf)
- Command Query Responsibility Segregation (CQRS) je šablon koji rešava ove probleme

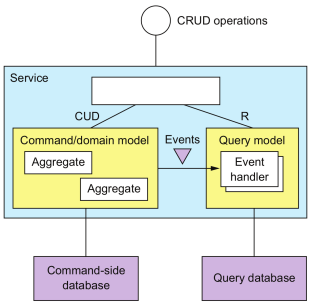
CQRS

- CQRS deli model podataka na onaj nad kojim obavljamo komande i onaj nad kojim obavljamo upite (može ih biti jedan ili više)
- Operacije kreiranja, ažuriranja i brisanja obavljaju se nad command modelom, dok upite formiramo nad query modelom
- Komande obavljamo nad command modelom, a zatim treba da sinhronizujemo query model
- Modul koji izvršava komande objavi događaj pri svakoj operaciji koju obavi, a query modul sluša te događaje i menja model shodno tome

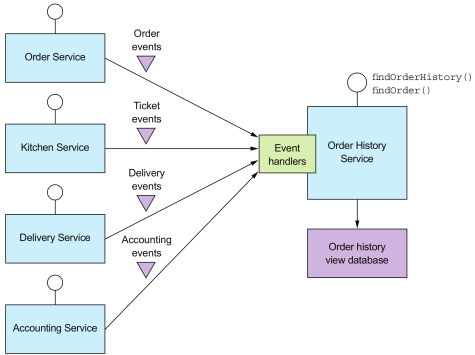


Podela command i query modula

- Mogu biti deo istog servisa



- A mogu biti i odvojeni servisi



Razmatranja

- Potrebno je odabrati odgovarajuće skladište podataka za svaki query model
- Moramo se nekako nositi sa tim što je query model eventually consistent (izvršene komande ne moraju biti odmah vidljive u query modelu)
- Event handler na query strani mora biti idempotentan (na primer pamtimo identifikatore događaja koji su obrađeni)

Primer

- *profile-service* podržava operacije kreiranja novog profila i izmenu ličnih podataka u okviru profila
- *follow-service* čuva informacije o praćenjima između profila i podržava kreiranje i brisanje praćenja, kao i dobavljanje svih pratilaca i praćenja nekog profila
- Kada hoćemo da prikažemo stranicu profila nekog korisnika, treba da prikažemo osnovne lične podatke tog korisnika, njegov broj praćenja i pratilaca
- *profile-service* sadrži ProfilePageView query model koji se ažurira CQRS šablonom
- Event handler sluša događaje *ProfileCreated*, *ProfileUpdated*, *FollowingCreated* i *FollowingDeleted* i shodno tome menja ProfilePageView

Zadaci

- API composition primer proširite tako da:
  - ordering servis kontaktira i catalogue servis, kako bi dobavio naziv, opis i boju svakog proizvoda iz porudžbine
  - izmenite OrderDetails strukturu tako da obuhvati i informacije o proizvodima
- CQRS primer proširite tako da:
  - following servis podrži upit koji za nekog korisnika istovremeno dobavlja liste imena svih njegovih pratilaca i praćenja
  - definišite adekvatan query model i sve događaje na koje event handler treba da reaguje