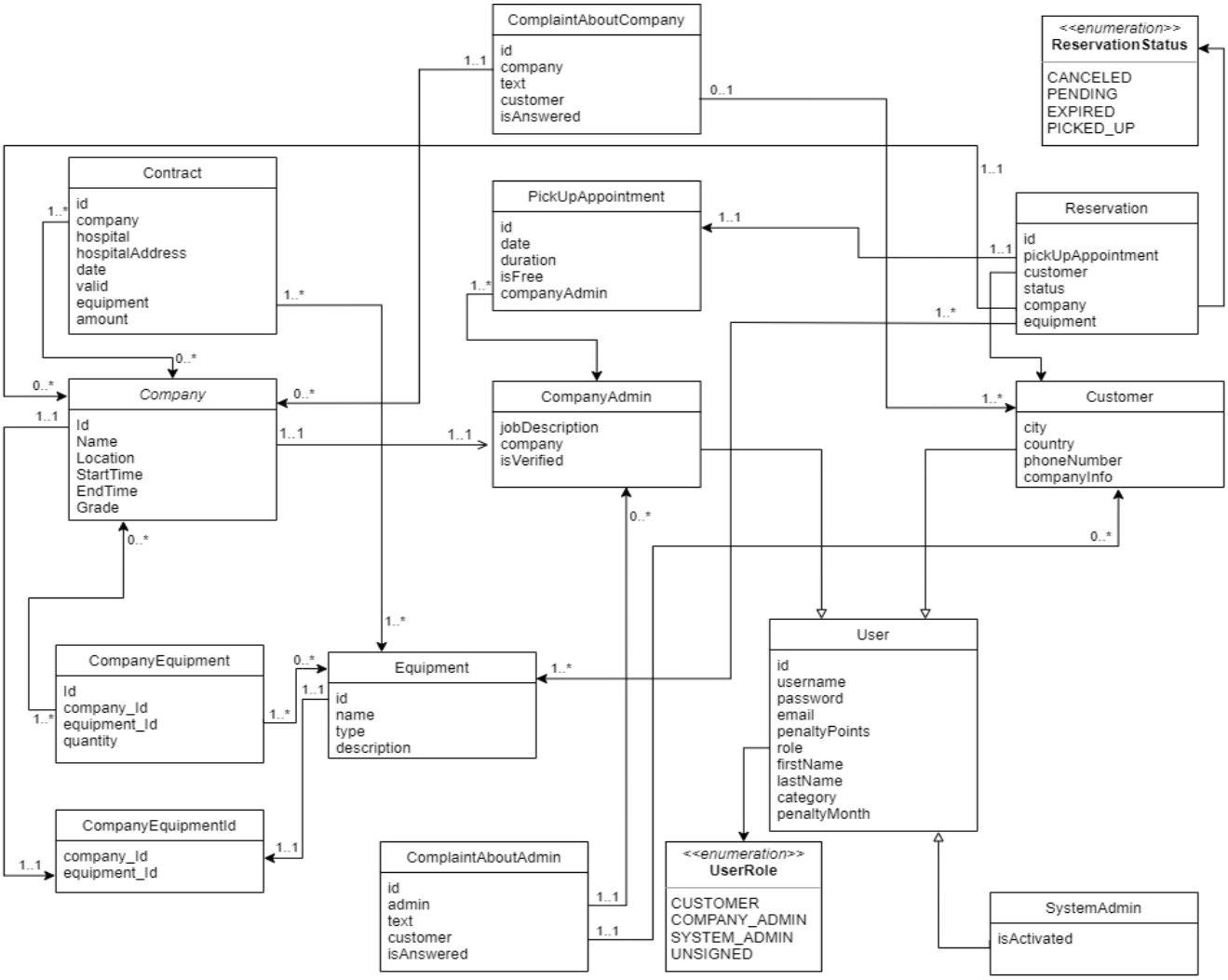
Proof Of Concept

Dizajn šeme baze podataka



Strategija za particionisanje podataka, replikaciju baze i obezbeđivanje otpornosti na greške

U cilju optimizacije performansi i povećanja otpornosti sistema, odlučili smo da bi dobra opcija predstavljala RAFT particionisanje za našu bazu podataka. RAFT je algoritam za distribuirano konzistentno replikovanje podataka na više servera koji obezbeđuje visok stepen dostupnosti i otpornost na kvarove.

Ovaj algoritam ima tri ključna dela: izbor vođe, replikacija zapisa i redosled upita.

1. **Izbor Vođe**
   - Kandidatura: Kada dođe do gubitka veze sa vođom ili prestanka rada vođe, serveri koji prepoznaju ovu situaciju postaju kandidati za novog vođu.
   - Glasanje: Kandidati zatim organizuju glasanje među serverima kako bi izabrali novog vođu. Serveri šalju glasove i razmatraju rezultate.
   - Većina Glasova: Novi vođa postaje onaj kandidat koji osvoji većinu glasova. Ovo garantuje da će najmanje polovina servera podržati novog vođu, održavajući time konsenzus.

2. **Replikacija Zapisa**: Replikacija zapisa osigurava da svi serveri u Raft klasteru imaju iste podatke. Ovaj proces se sprovodi na sledeći način:
   - Pisanje Zapisa: Kada klijent uputi zahtev za pisanje, vođa prima zahtev, beleži ga i prosleđuje ga ostalim serverima.
   - Potvrda Većine: Zapisi se smatraju validnim tek kada većina servera potvrdi da su ih primili. Ovo osigurava doslednost između svih replika.
   - Slučaj Gubitka Vođe: Ako dođe do gubitka vođe, novi vođa preuzima ulogu i nastavlja sa replikacijom. Ako stari vođa ponovo postane dostupan, on se prilagođava novom vođi kako bi održao doslednost.

3. **Redosled Upita**: Raft algoritam obezbeđuje redosled izvršavanja upita kako bi održao doslednost između svih servera. Ovo se postiže na sledeći način:
   - Sekvenciranje Zapisa: Svaki zahtev za pisanje ima jedinstveni identifikator sekvence, koji se koristi za obezbeđivanje redosleda izvršavanja.
   - Vođa koordinacija: Vođa koordiniše redosled izvršavanja upita, obezbeđujući da se zahtevi primenjuju u odgovarajućem redosledu na svim serverima.
   - Distribucija Redosleda: Odluke o redosledu upita distribuiraju se na sve servere, održavajući doslednost širom particionisane baze podataka.

Razmotrićemo strategiju podela baze podataka na više servera kako bismo postigli horizontalno skaliranje i bolju raspodelu opterećenja. Definisaćemo koje podatke ćemo razdeliti između servera i kako ćemo organizovati šeme baze podataka na svakom serveru (eventualno bismo mogli sve podatke šeme baze podataka raspodeliti na sve servere).

Strategija za keširanje podataka (Redis)

U cilju poboljšanja performansi i efikasnijeg upravljanja podacima u našem projektu, predlažemo uvođenje Redis-a kao eksternog keša. Redis će poslužiti kao brz i efikasan način čuvanja i pristupa podacima koji se retko menjaju, posebno u kontekstu informacija o kompanijama i preparatima u sistemu.

Motivacija za uvođenje Redis-a:

1. **Brzina pristupa podacima**: Redis je in-memory baza podataka što omogućava brz pristup podacima. Ovo je od suštinskog značaja za informacije koje se često čitaju, kao što su podaci o apotekama i preparatima.
2. **Lakoća implementacije**: Redisson biblioteka olakšava integraciju Redis-a u Java projekat. Implementacija je jednostavna, a Redisson pruža programski interfejs koji olakšava rad sa Redis operacijama.
3. **Skalabilnost**: Redis podržava horizontalno skaliranje, što znači da možemo dodavati više Redis instanci kako bi se povećala sposobnost keširanja i bolje nosila sa povećanim opterećenjem.
4. **Efikasno keširanje retko ažuriranih podataka**: Za podatke koji se retko menjaju, poput informacija o kompanijama i njihovim preparatima, Redis je optimalan. Keširanje ovih podataka u Redis-u umanjuje potrebu za čestim pristupima bazi podataka i povećava brzinu odgovora.

Kada klijent pokrene zahtev koji uključuje čitanje određenih podataka, aplikacija prvo proverava da li su ti podaci već prisutni u Redis kešu. Ako su prisutni, podaci se dobijaju iz keša, što omogućava brz pristup. Ako podaci nisu u kešu ili su istekli, aplikacija ide do osnovne baze podataka, a zatim ažurira keš sa novim podacima koje je dobila. Ovaj pristup omogućava dinamično keširanje i osvežavanje podataka, čime se postiže balans između brzog odgovora na česte zahteve i osvežavanja podataka kako bi bili tačni i ažurni.

Okvirna procena za hardverske resurse potrebne za skladištenje svih podataka u narednih 5 godina

Entitet skladištenja	Memorija skladištenja entiteta	Memorija veće količine entiteta
User	225B	88.5GB (100 miliona korisnika)
Reservation	177B	84.5MB (500 000 rezervacija na mesečnom nivou)

Predlog strategije za postavljanje load balansera

Predlažemo integraciju Nginx-a sa Raft serverima kako bismo postigli robustnu i visoko dostupnu arhitekturu za našu aplikaciju. Ova arhitektura će obezbediti efikasno rukovanje zahtevima, otpornost na kvarove i jednostavno skaliranje.

Nginx će se koristiti kao centralni element arhitekture, preuzimajući uloge web servera, obrtno-proxy servera i servera za ravnotežu opterećenja. Primaće zahteve od klijenata i distribuirati ih među više Raft servera koji će činiti backend klaster.

Raft serveri će biti odgovorni za skladištenje podataka i održavanje doslednosti između sebe. Korišćenjem Raft particionisanja, omogućićemo visok stepen dostupnosti i automatsko preuzimanje u slučaju kvara nekog od servera. Svaki Raft server sadržavaće kopiju podataka i raditi sinhronizaciju sa ostalima.

Nginx će takođe delovati kao obrtno-proxy, čime ćemo skriti kompleksnost backend infrastrukture od klijenata. Ovo će unaprediti bezbednost sistema i omogućiti efikasno upravljanje zahtevima. SSL/TLS terminacija će biti implementirana na Nginx-u, što će dodatno poboljšati performanse backend servera oslobađanjem njih od tereta enkripcije.

Arhitektura će omogućiti efikasno skaliranje, ravnomerno raspodeljivanje opterećenja i jednostavno održavanje sistema. Uz korišćenje Nginx-a i Raft servera, postići ćemo arhitekturu koja zadovoljava potrebe naše aplikacije u pogledu performansi, dostupnosti i otpornosti na greške.

Predlog koje operacije korisnika treba nadgledati u cilju poboljšanja sistema

Cilj praćenja operacija korisnika je poboljšanje sistema putem personalizacije usluga i pružanje relevantnih preporuka. Predložene operacije za nadgledanje korisničkog ponašanja su ključne za stvaranje boljeg korisničkog iskustva i povećanje angažovanosti.

Praćenje istorije pretrage korisnika ima za cilj razumevanje njegovih interesa. Analiza često pretraživanih pojmova i kategorija proizvoda pomaže u stvaranju personalizovanog profila korisnika. Takođe, praćenje preferenci kompanija omogućava nam da utvrdimo koje kompanije su često odabrane ili označene kao omiljene, kao i da pratimo učestalost kupovine određenih kompanija.

Ključni korak ka personalizaciji korisničkog iskustva je razvoj algoritma za personalizovane preporuke. Na osnovu istorije pretrage i preferenci kompanija, ovaj algoritam bi mogao predlagati nove proizvode ili kompanije koje bi mogle biti interesantne korisniku. Ovo bi značajno unapredilo korisničko iskustvo, čineći ponudu proizvoda i usluga bližom individualnim potrebama.

Dodatno, predlozi za kupovinu i poručivanje mogu se implementirati kao deo personalizovane usluge. Na osnovu praćenja pretraga, preferenci kompanija i istorije kupovine, sistem bi mogao predlagati proizvode koji su u skladu sa korisnikovim interesovanjima. Ovaj pristup ne samo da povećava korisničko zadovoljstvo već i podstiče dodatnu potrošnju.

Kompletan crtež dizajna predložene arhitekture (aplikativni serveri, serveri baza, serveri za keširanje, itd)