Skalabilnost

1. Dizajn šeme baze podataka (konceptualni, logički ili fizički)

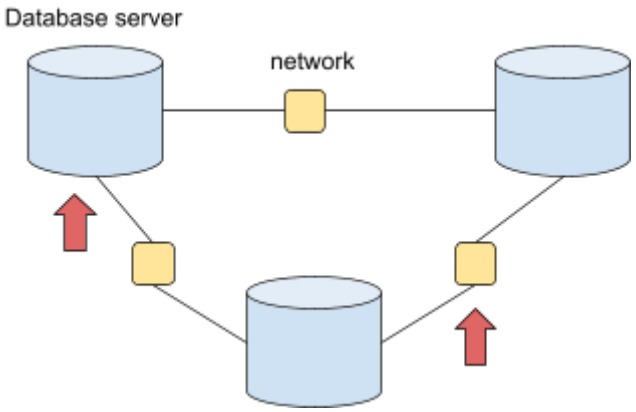
Dizajn šeme baze podataka se nalazi u folderu kao slika, zbog preglednosti podataka.

2. Predlog strategije za particionisanje podataka

Kako bi smanjili zahteve za resursima kompleksnih upita particionišemo podatke. Horizontalnim particionisanjem možemo podeliti tabele koje će potencijalno imati veliki broj redova kao što su rezervacije lekova, zakazani pregledi i konsultacije, na više tabela sa manjim brojem redova koje će grupisati podatke po nekom ključu. Za ključ horizontalno particionirane tabele možemo uzeti apoteku ili skup po proceni potencijalnog broja njenih korisnika. Možemo podeliti tabelu appointment na dve gde će ključ biti tip appointmenta: konsultacija ili pregled. Vertikalnim particionisanjem možemo izvući atribute entiteta za koje korisnici retko prave upit, kako bi oslobodili kvalitetniji hardware čestim upitima. Iz appointment tabele možemo izvući trenutak početka i kraja kako bi ubrzali zahteve ka slobodnim terminima, pošto used čestih upisa ovaj deo ne možemo keširati. Takođe možemo izvući deo sa lozinkama kako bi povećali bezbednost aplikacije.

3. Predlog strategije za replikaciju baze i obezbeđivanje otpornosti na greške

Pošto se sistem većinski koristi za rezervacije, očekuje se veliki broj write naredbi, tako da klasterujemo relacione baze podataka u multiple master arhitekturi. Svaki database server obrađuje update i insert upite i iste prosleđuje ostalim database serverima kako bi replikacije bile u konzistentnom stanju. Potrebno je da imamo više od dva database servera da usled otkaza mreže ne bi došlo do skladištenja različitih podataka, kako je svaki master. Jedini događaj koji bi doveo replikacije u ne konzistetno stanje u arhitekturi na slici je otkaz database servera i mreže između druga dva dabase servera, što procenjujemo kao veoma malu verovatnoću. U slučaju da se poveća broj read upita možemo dodati dodatni slave database server sa koga se samo čitaju podaci. Transackije i optimističko zaključavanje



4. Predlog strategije za keširanje podataka

Podatke o korisnicima, zaposlenima, apotekama, lekovima možemo keširati kako ne očekujemo česte write, update nardbe, u slučaju do njih možemo updatovati cache. Za statistiku, srednju ocenu nećemo praviti uvek novi upit kako očekujemo da još jedan novi podatak u čitavoj bazi ne može uticati na rezultat, tako da ove podatke možemo updatovati u nekom vremenskom intervalu na cachu. Upite za koje očekujemo česte promene rezultata usled čestih write naredbi, rezervacija leka, upit za slobodnim terminima, možemo podržati least recently used cache strategijom gde ćemo keširati rezultate za poslednje upite. Jedan od najpopularnijih nacina za kesiranje podataka u Spring-u je koriscenje EhCache. EhCache je open-source standardno-baziran kes koji povecava performanse, rasterecuje bazu podataka i pojednostavljuje skalabilnost.Najcesce se implementira samo dodavanjem anotacija. Njegovom upotrebom bi se rasteretila potreba za stalnim pristupom bazi podataka koja bi dodatno usporila aplikaciju za podatke koji se retko menjaju. U nasem slucaju najbitnije bi bilo kesirati podatke vezane za profil apoteka, pristup lekovima, pristup predefinisanim pregledima.

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

Uočili smo entitete koji se registruju samo jednom u okviru sistema i čiji broj nije dovoljno velik tako da utiče na potrebno skladište u dužem periodu na ovako veliki broj korisnika. Entiteti uzeti u obzir su nalozi korisnika, pregledi, izveštaj sa pregleda, rezervacije lekova, i ocene i žalbe. Sa ovim uzetim u obzir ako je ukpan broj korisnika 200 miliona i perzistencija korisnika u bazi zauzima 0.5kb po korisniku što čini približno 95.6gb. Od procenjenih milion pregleda i rezervacija leka, uzećemo da na svaku devetu rezervaciju leka će biti jedan pregled. Za perzistenciju jedne rezervacije leka u proseku je potrebno 0.15kb što u narednih pet godina čini 7.72gb. Za perzistenciju jednog pregleda je u proseku potrebno 0.2kb. Ukoliko na mesečnom nivou imamo 100.000 pregleda, nakon pet godina za preglede će biti potrebno 1.14gb. Procenjujemo da će 5% korisnika otkazati pregled ili se neće pojaviti na istom. Procenjuje se da će mesečno biti 95.000 izveštaja što nakon pet godina čini približno 8.15gb. U proseku ocena ili komentar zauzimaju 0.4kb što u slučaju da 10% korisnika ostavi ukpno će zauzeti 250mb. Perzistencija svih ostalih entiteta će biti sigurno manja od ocena i komentara, računaćemo da će u proseku uzimati 30mb što za ostalih 40 entiteta čini 1.2gb. Tako da ukupno procenjujemo da će u narednih pet godina biti potrebno 114gb za skladištenje svih podataka.

6. Predlog strategije za postavljanje load balansera

Iskoristili bi smo weighted round robin algoritam pod pretpostavkom da nemamo servere podjednako snaznih konfiguracija ili planiramo u buducnosti da imamo servere sa razlicitim konfiguracijama u nasem klasteru. Zahtevi bi se rasporedjivali ciklicno po cvorovima uzimajuci u obzir i tezine cvorova. Jace konfiguracije bi imale vecu tezinu i primale bi vise zahteva.
U nasem slucaju radi se o stateless web aplikaciji.

7. Predlog koje operacije korisnika treba nadgledati u cilju poboljšanja sistema

Implementacijom event sourcing mehanizma imaćemo sve potrebne podatke za implementaciju bilo koje statistike koju će poslovanje tražiti od nas. Što tehničkog dela rešenja, njegove upotrebljivosti tako i ponašanja korisnika i njihovih interesovanja. Na ovaj način možemo pratiti potražnju za lekove i na taj način obezbediti da apoteke imaju uvek tačnu informaciju za nove nabavke. Kao i broj poseta farmaceutu ili dermatologu na koji način se može pratiti kvalitet njihove usluge.

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