PHARMACY
PROOF OF CONCEPT

Tim 9 – SIIT

Isidora Stanić - SW-16/2018
Miloš Popović - SW-24/2018
Milica Đumić - SW-27/2018
Mladen Vasić - SW-28/2018

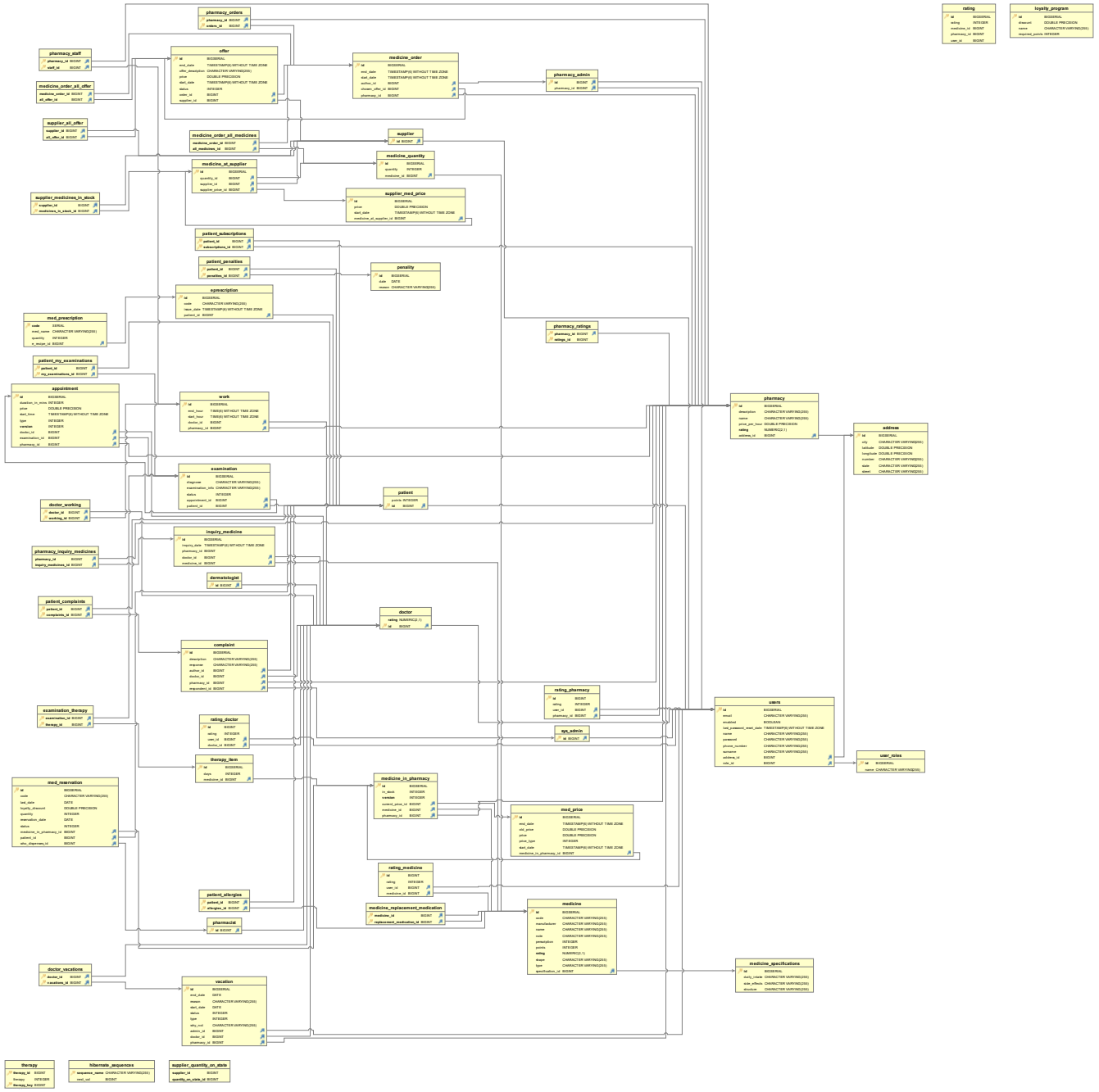
Zahtevi

Trenutna verzija aplikacije može da uspešno opsluži manji broj korisnika, trenutno je testirano samo na desetinama korisnika, a očekuje se da je ograničenje oko 1000 korisnika (oko 100 dnevno).

Zahtev ovog projekta podrazumeva da aplikacija podrži 200 miliona korisnika, sa milion mesečnih rezervacija, konsultacija i pregleda.

1. Baza podataka

Er diagram baze



2. Particionisanje baze

Trenutno aplikacija koristi jednu POSTGRES SQL bazu za čuvanje svih podataka, međutim jedna baza ne može fizički da se skalira na ogroman broj korisnika i dolazi će do preopterećenja. Svakako održavanje toliko zahtevne baze može biti skupo.

Da bi smo rasteretili bazu možemo uraditi vertikalno (po kolonama) ili horizonalno (po redovima) particionisanje. Kako se ovde radi sa podacima koji su aktuelni samo u jednom periodu a najčešće nam trebati sve informacije za svaki entitet, te predlažemo horizontalno particionisanje.

Particionisanje tabele korisnika

Tabela od 200 miliona korisnika se mora particionisati zbog brzine pristupa.

Pošto se najčešće korisnicima pristupa pomoću username-a, tabelu korisnika možemo podeliti na više tabela, a algoritam raspodele može biti hash funkcija po username-u.

Problem sa ogromnim brojem korisnika je taj što se ogroman broj korinsika ne pojavljuje nikada u sistemu, pa bi korisno da se neaktivni korisnici prebace u posebnu tabelu, da ne bi smetali u pretrazi.

Particionisanje tabela pregleda i rezervacija

Ove tabele su jedne od najkritičnijih tabela u sistemu pošto će se najviše entiteta dodavati i preuzimati iz istih.

Potrebno je napraviti više baza podataka, kojima će se pristupati na osnovu nekog algoritma za particionisanje, koji može biti jednostavan moduo po ID-ju entiteta po N(broj particionisanih baza). Smatramo da nema potrebe za komplikovanijom raspodelom, pošto su su samo noviji entiteti više favorizovani (budući pregledi, rezervacije u budućnosti) i češće će se dobavljati pa će podela po modulu odraditi korektan balans.

Sa druge strane može se nakupiti mnogo zastarelih podataka o rezervacijama i pregledima koje nam u velikom broju slučajeva neće trebati osim kao pregled istorija nekih pacijenata. Ideja je da se ti podaci izmeste u posebnu tabelu svakog meseca.

3. Replikacija baze podataka

Pored particionisanja baze potrebno je dodatno optimizovati čitanje baze replikacijom podataka, opredelili smo se za Master-Slave replikaciju zbog konzistentnosti pošto su podaci u ovakvim aplikacijama veoma bitni.

Što se tiče sinhornizacije replikacije, opredelili smo se za sinhronu sinhronizaciju baze takođe zbog važnosti podataka kako bi smanjili rizik za izgubljenim transakcijama. Sa druge strane ukoliko dođe do otkaza neke replike baze, ne narušava se konzistentnost podataka.

Prednost sinhornizacije svih master baza je ta što možemo čitanje vršiti iz bilo koje replike baze i biti signurni da su svi podaci autentični. Što doprinosi brzini same baze.

4. Predlog strategije za keširanje podataka

Keširanje pomoću hibernate

Kako je L1 već podržan od hibernate-a, možemo optimizovati korišćenjem L2 keša pomoću EhCahce provajdera. Informacije o lekovima i apotekama ćemo keširati READ WRITE strategijom, eventualno ukoliko

smo sigurni da se informacije o leku neće menjati možemo dodatno optimizovati korišćenjem Nonrestricted Read Write strategijom.

Informacije o korisnicima je takođe potrebno keširati pošto će se ti podaci retko, a možda i nikada neće menjati.

Pošto će svaki korisnik moći da pregleda termine dobro bi bilo keširati iste, kako ne bi za svakog korisnika opterećivali bazu, a ukoliko korisnik rezerviše onda moramo dobaviti iz baze objekat, kako bi bili uvereni da podatak nije u međuvremenu izmenjen.

Keširanje statičkih podataka

Kako je aplikacija struktuirana kao REST servis, frontend nam je potpuno statičan i možemo koristiti CDN za keširanje statičkog sadržaja. Ovim smo drastično rasteretili server za frontend. Takođe neki statički podaci kao slike apoteka, doktora se mogu takođe keširati uz pomoć CDN-a.

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

Tabela korisnika

Prosečna veličina jednog reda u tabeli korisnika je 1.6kB što ako se skalira na 5 godina i dođemo do 200M korisnika konačna vrednost baze bi iznosila 320GB

Tabela pregleda i termina i Rezervacija leka

Jedan pregled sa osnovnim informacijama i terapijom zauzima u proseku 1KB.

Jedna rezervacija leka u bazi prosečno zauzima 0.3KB.

Kada skaliramo te podatke na 5 godina, sa pretpostavkom da će biti 1 milion novi rezervacija i zakazanih termina mesečno dobijamo 1.3KB* 1M * 12 * 5 = 78GB

Ostalo

Skalirana pretpostavka za sve ostale podatke iznosi oko 4kB po korisniku, analogno na 200 miliona korisnika ostali podaci zauzimaju 800GB

Ukupno

Konačna pretpostavka za veličinu skladiša tokom 5 godina korišćenja aplikacije iznosi ~1.2TB

Veličina memorije je računata pomoću metode

```
SELECT pg_size_pretty( pg_total_relation_size('table_name') );
```

Dobijena vrednost je podeljena sa brojem redova da bi dobili prosečnu veličinu.

6. Predlog strategije za postavljanje Load Balansera

Za backend ovog porjekta je korišćena je java platforma uz pomoć Spring Boot Frameworka koji se oslanja na Apache Tomcat server.

Tačkama 2, 3 i 4 smo opisali kako bi optimizovali bazu podataka, ali sa 200M korisnika i 1M novih entiteta svaki mesec, sam backend spring server postaje usko grlo i moramo ga optimizovati.

Slično kao kod baze na backend serveru imamo hardversko ograničenje koliko možemo da unapredimo performanse, ali i sa druge strane Tomcat nas ograničava na 200 thredova u aplikaciji.

Rešenje koje ćemo primeniti je horizontalno skaliranje aplikacije odnosno dodajemo više backend servera kojima se pristupa preko Load Balansera. Ovim smo rasteretili zahteve korisnika na više servera koje reguliše load balanser po Round Robin principu. Pošto nam je backend Stateless REST aplikacija i nismo vezani za sesiju korisnika, balansiranje sadržaja plan je i vertikalno skalirati aplikaciju, odnosno podeliti je na mikro servise.

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

Što se tiče iz bezbedonosnih razloga neophodno je logovati sve akcije korisnika, ukoliko dođe do neočekivanih rezultata u izveštajima da bi mogli ispratiti dešavanje u sistemu.

Što se tiče korinsičkog iskustva pratićemo koje apoteke korisnik najčešće posećuje, koje lekove najčešće uzima kako bi ih izdvojili kao preporučene.

Moguće je pratiti kretanje miša pomoću mouseflow plugina kako bi pratili intuitivnost aplikacije, i shvatili da li se korisnici lagano snalaze u aplikaciji.

Korisno bi bilo pratiti koje akcije prave regularni korisnici i na osnovu tih podataka po potrebi promeniti način na koji particionišemo podatke ukoliko zaključimo da bi takva promena bila isplativa.

8. Predlog arhitekture