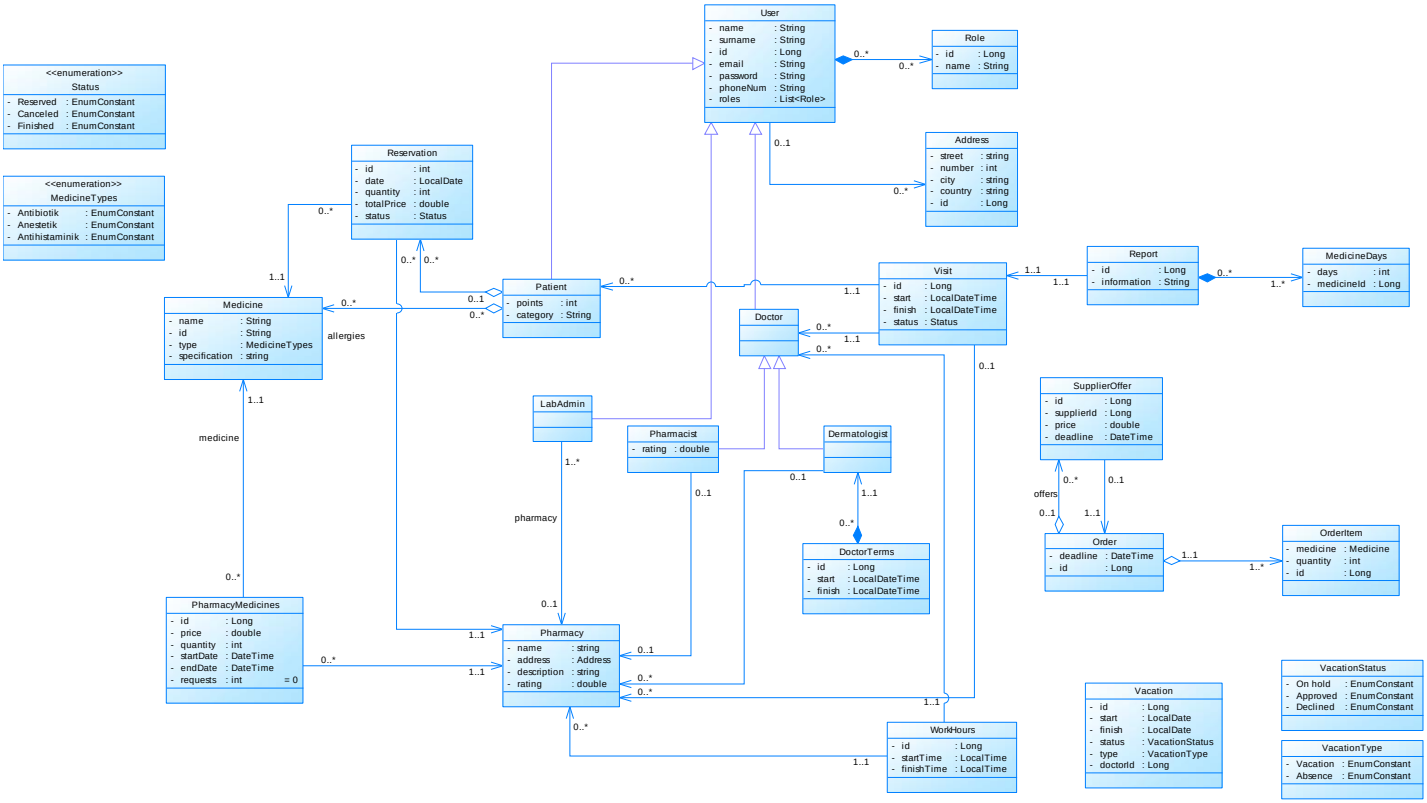
Proof of concept

Dizajn šeme baze

Dizajn šeme baze predstavljamo kroz klasni dijagram. Kako svaka od navedenih klasa sadrži Entity odnosno Embedded notaciju, relacije se automatski kreiraju.



Predlog strategije za particionisanje podataka

Kako tabele mogu biti brzo popunjene podacima, čitanje podataka može postati skupa operacija sa stanovišta procesorskog vremena i potrošnje memorije. Kako bi se umanjio ovaj problem, odnosno ubrzao rad sa tabelama potrebno je podeliti određene tabele na particije. Na ovaj način pojedinačne operacije su efikasnije ali je pretraživanje cele tabele manje efikasno.

Relacije koje je potrebno i smisleno podeliti na particije su:

- Orders

  Tabelu je moguće podeliti na dve particije. Jednu za one porudžbine čiji *deadline* nije prošao i drugu za one gde jeste. Na ovaj način ne gubimo (brisanjem) istekle porudžbine, koje koristimo u analitičke svrhe, a smanjujemo broj porudžbina koje će se učitavati prilikom selekcija.

- Reservations

Jedna od najzahtevnijih tabela u sistemu je upravo tabela u koju čuvamo podatke o rezervacijama lekova. Kada bismo je podelili na tri particije u zavisnosti od statusa rezervacije leka, dobili bismo ubrzan pristup svakoj kategoriji. Na primer, prilikom izdavanja lekova jedan upit farmaceuta bi se izvršio samo nad rezervacijama sa statusom *rezervisan*, a ta particija bi bila mnogo manja od particije u kojoj bismo čuvali rezervacije sa statusom *izdat*. Takođe prilikom učitavanja podataka za izveštaje ne bi se učitavali rezervisani i otkazani lekovi jer oni ne utiču na zaradu apoteke, već bi na brže dobili samo izdate lekove.

Statusi rezervacije:

1. Rezervisan
2. Otkazan
3. Izdat

- Visits

Kao i rezervacije, tabela za čuvanje podataka o pregledima i savetovanjima je veoma zahtevna tabela (procena je milion redova mesečno). Kao i kod rezervacija, upiti nad ovom relacijom se često svode na samo aktivne ili samo obavljene, tako da ima smisla podeliti particije isto kao i kod rezervacija. Statusi su jednaki kao i kod rezervacija.

- Users

Očekivani broj korisnika aplikacije je 200 miliona. To je velik broj podataka za učitavanje sa znanjem da se dosta podataka o korisniku neće koristiti. Bilo bi poželjno izvršiti vertikalnu podelu tabele, tako da u jednoj tabeli ostavimo podatke koji se koriste za prijavu na sistem (email i lozinka) a ostale podatke grupišemo u novu tabelu.

Predlog strategije za replikaciju baze i obezbeđivanje otpornosti na greške

Replikacija predstavlja udvajanje komponenti računarskog sistema. Replikacije ćemo koristiti isključivo zbog pouzdanosti (nećemo vršiti replikacije zbog performansi). Za način replikacije, odabrali smo ROWA*(Read One Write All)* protokol. Koncept ovog protokola je da se samo primarna kopija čita, a sve menjaju. Konkretno koristićemo ROWA protokol sa tokenima pravih kopija. Ovaj protokol se razlikuje od osnovnog ROWA protokola po tome što svaki podatak ima jedan ekskluzivan token ili skup deljivih tokena. Kada kopija mora da izvrši operaciju pisanja, locira i pristupa ekskluzivnom tokenu, ukoliko ne postoji locira sve neispravne tokene i pravi novi ekskluzivan token. Kada kopija mora da izvrši operaciju čitanja, locira deljivi token, kopira njegovu vrednost, pravi i čuva novi deljivi token. Ako deljivi token ne postoji, kopija ga locira i konvertuje ga u deljivi. Svaki podatak ima ili ekskluzivan token ili skup deljivih tokena. Operacija pisanja mora da dobije ekskluzivan dok operacija čitanja mora da dobije bar deljivi token.

Predlog strategije za keširanje podataka

Keširanje predstavlja smeštanje podataka zarad bržeg budućeg pristupa istim podacima. Kako Spring podržava keširanje i omogućuje laku realizaciju (upotrebom anotacija Cacheable, CacheEvict, CachePut...) potrebno je odabrati koji će se podaci keširati. Kako se keš brzo popunjava treba obratiti pažnju kada ga popunjavati a kada prazniti. Kada bismo ispratili jedan standardni use case znali bismo kada da keširamo koje podatke. Anotaciju @Cacheable bismo koristili na metodama koje dobavljaju mali broj podataka iz velikog skupa podataka. To su npr.

- Prikaz pregledanih pacijenata za doktora (Cacheable("patients"))
- Prikaz zakazanih pregleda za pacijenta (Cacheable("scheduled"))
- Informacije o pregledu za doktora (Cacheable("visit-info"))
- Informacije o profilu za sve korisnike (Cacheable("profile"))
- Rejtinzi apoteka (Cacheable("rating"))

Oslobađanje keša bismo radili CacheEvict anotacijom. Za vrednost *value* bismo prosledili one delove za koje sumnjamo da će biti potrebni u skorijem korišćenju. Na primer kada doktor započinje pregled možemo osloboditi sav keš izuzev informacija o profilu pacijenta(pogodno zbog bržeg učitavanja alergija). Takođe Spring podržava uslovno keširanje koje se definiše u parametru *condition*. Na ovaj način možemo keširati različite podatke u zavisnosti od uloge korisnika, količine podataka i slično.

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

Računanje obavljamo uzimajući u obzir vrednosti iz tabele.

Tip	Zauzeće memorije(byte)
INT	4
BIGINT	8
LONG	8
ENUM	1
VARCHAR(255)	256
TIMESTAMP	4
DATE	3
TIME	3
DOUBLE	8

- Korisnici
  $x = 8*2 + 5*256$
  Dobijamo rezultat da nam je za jedan red u tabeli *users* potrebno 1,26KB.
  Da bismo zadovoljili potrebe od 200 miliona korisnika potrebno nam je 252GB.
- Rezervacije

x = 8*8
Dobijamo rezultat da nam je za jedan red u tabeli *reservations* potrebno 64B.
Da bismo zadovoljili potrebe da mesečno imamo milion sačuvanih rezervacija, kroz pet godina potrebno nam je 3,84GB.

- Posete
  x = 4*8 + 3*4
  Dobijamo rezultat da nam je za jedan red u tabeli *visits* potrebno 44B. Da bismo zadovoljili potrebe da mesečno imamo milion sačuvanih poseta, kroz pet godina potrebno nam je 2,64GB.

- Izveštaji o pregledu
  x = 2*8 + 256 + 5*20
  Računamo da je prosečan broj prepisanih lekova za jedan pregled 5 (koristi se pomoćna tabela *medicine_days* u kojoj jedan red zauzima 20B). Dobijamo rezultat da nam je za jedan red u tabeli *reports* potrebno 372B. Da bismo zadovoljili potrebe da mesečno imamo milion izveštaja, kroz pet godina potrebno nam je 22,32GB.

- Apoteke
  x = 4*8 + 2*256
  Dobijamo rezultat da nam je za jedan red u tabeli *pharmacies* potrebno 544B. Da bismo zadovoljili potrebe da u sistemu imamo 1.000.000 apoteka, potrebno nam je 0,544GB.

- Lekovi
  x = 4 + 8 + 2*256
  Dobijamo rezultat da nam je za jedan red u tabeli *medicines* potrebno 524B. Da bismo zadovoljili potrebe da u sistemu imamo sve moguće lekove 14528 (izvor go.drugbank.com/stats) potrebno nam je 7,79 MB, što je jako malo u odnosu na ostale tabele, ali kako imamo tabelu koja povezuje lekove i apoteke potrebno je videti koliko prostora ona zahteva.
  x = 7*8
  Dobijamo rezultat da nam je za jedan red u tabeli *pharmacy_medicines* potrebno 56B.
  Da bismo zadovoljili potrebe da u svakoj apoteci u sistemu imamo sve moguće lekove, potrebno nam je 831GB.

Gore navedeni podaci su oni koji se čuvaju za stalno, tj. ne brišu se dinamički. Kako se podaci poput slobodnih termina kod dermatologa brišu rezervacijom ili po njihovom isteku nećemo ih računati jer ne mogu uticati na ukupno zauzeće u dovoljnoj meri.

Kada saberemo dobijene vrednosti, saznajemo da nam je potrebno 1.1TB za potrebe sistema.

Predlog strategije za postavljanje load balansera

Load balanser predstavlja raspodelu zadataka preko skupa resursa sa ciljem da njihova ukupna obrada bude efikasnija. Load balanser prihvata zahteve a zatim ih distribuira na servere koji su dostupni za određene zahteve. Predlog algoritma za SLB je Least Connection algoritam. Least Connection, kako mu ime kaže prosleđuje klijentov zahtev do servera sa najmanjim brojem aktivnih konekcija u trenutku primanja zahteva. Ukoliko su serveri različiti (sa staništa hardverskih mogućnosti) moguće je opredeliti se za Weighted Least Connection algoritam, koji pored aktivnih konekcija, uzima u obzir i "težinu" svakog servera. Težinu definiše administrator po odabranom kriterujumu.

Predlog koje operacije korisnika treba nadgledati u cilju poboljšanja sistema

Kako bi se sistem dodatno unapredio, poželjno je nadgledati operacije svih tipova korisnika. Najznačajnije su operacije koje su zahtevnije sa stanovišta korisnika (gde je komunikacija sa interfejsom zahtevnija). Npr. izdavanje leka (farmaceut) je mnogo manje zahtevna operacija od zakazivanja novog termina za pacijenta (farmaceut ili dermatolog). Značajna poboljšanja bi bila automatsko popunjavanje formi, autocomplete i postavljanje *default* vrednosti na polja kao što su izbor datuma i vremena. Operacije za koje smatramo da je potrebno nadgledati ih su:

- Zakazivanje novog pregleda/savetovanja (pacijent, dermatolog, farmaceut)
- Rezervisanje leka (pacijent)
- Praćenje alergija (dodavanje i brisanje) (pacijent)
- Prepisivanje terapije (farmaceut, dermatolog)
- Dodavanje novih zaposlenih u apoteku (admin apoteke)
- Prikaz izveštaja za zadati vremenski interval (admin apoteke)

Crtež dizajna predložene arhitekture