Univerzitet u Novom Sadu
Fakultet tehničkih nauka Novi Sad
Departman za računarstvo i automatiku

Blood Bank - proof of concept

Internet Softverske arhitekture

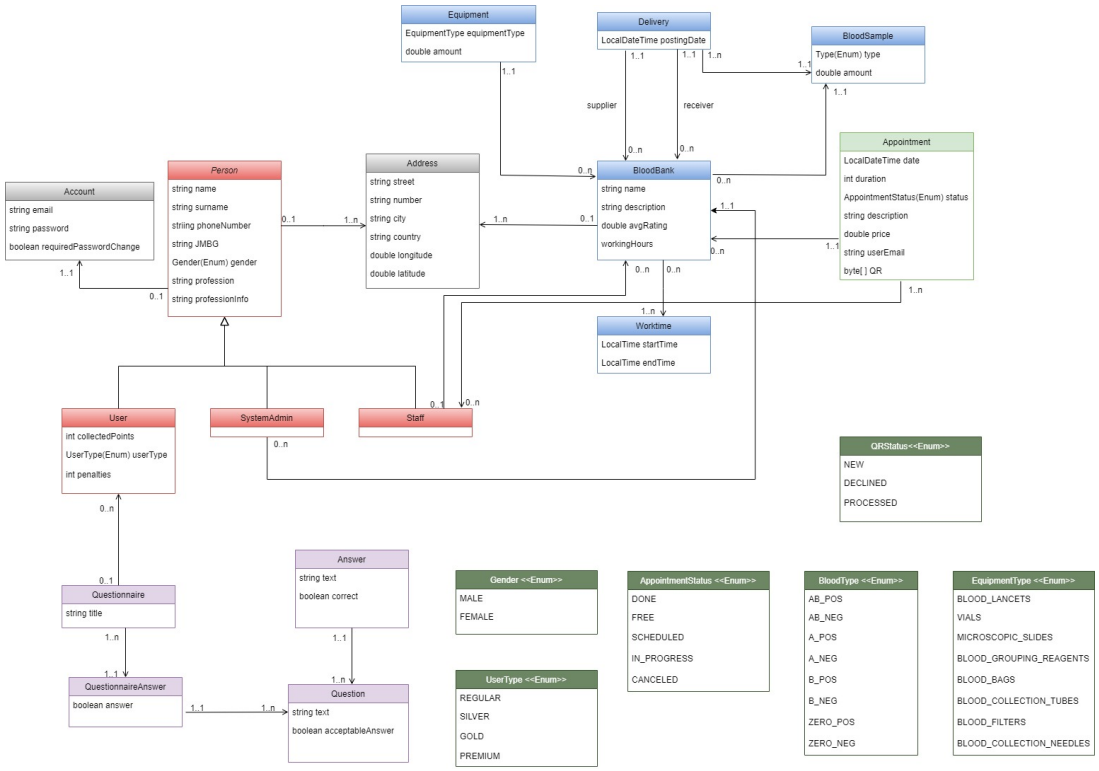
Studenti:
Miloš Zeljko RA24/2019
Mihailo Veljić RA32/2019
Nikola Holjevac RA11/2019
Milena Jelić RA23/2019

Novi Sad,
Januar 2023.

Sadržaj

1 Dizajn šeme baze podataka

Klasni dijagram predstavljen na slici ispod prestavlja dizajn šeme baze podataka. Korišćen je PostreSQL sistem za upravljanje bazom podataka, a sama baza je kreirana posredstvom Spring Boot aplikacije.



Slika 1: Konceptualni model baze podataka

2 Predlog strategije za particionisanje podataka

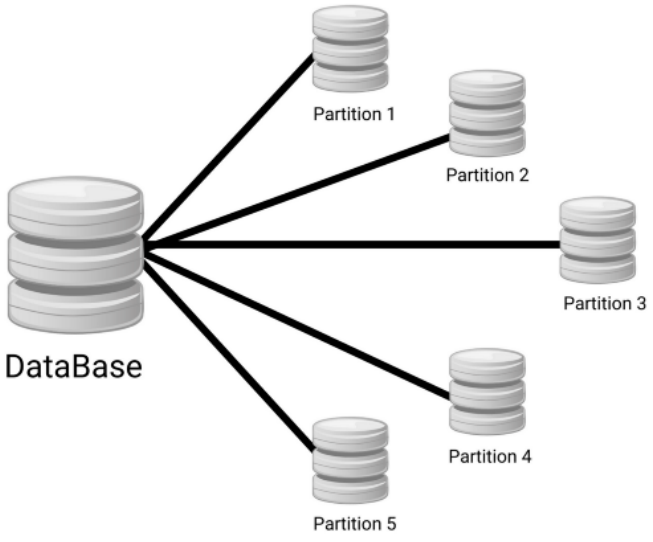
Kako će vremenom broj klijenata porasti, pojaviće se potreba za skalabilnošću, odnosno čuvanjem podataka na više servera. Radi brzog i efiksnog dobavljanja podataka, vrlo je važno smisleno particionisati podatke po serverima, kako bi se poboljšala efikasnost upita i održavanje podataka.

Kako je jedna od glavnih funkcionalnosti ove aplikacije rad sa terminima, ima smisla particionisanje vršiti po njima.

- Prvi pristup može biti čuvanje po datumu kreiranja i/ili održavanja, gde je smisleno da skoriji termini budu lakše dostupni, dok su prošli termini uglavnom nisu toliko bitni, tek možda za neke statistike, pa je potrebno uložiti manje resursa za njihovo čuvanje.

- Drugi pristup može biti kreiranje particija po terminima vezanim za jednog člana osoblja. U ovom slučaju bi se prilikom zahteva za čitanje/pisanje termina slao id osoblja na osnovu koga bismo znali kojoj particiji da pristupamo. Problem može biti ukoliko neki pacijent dobavlja svoje termine, a oni nisu od istih lekara, biće potrebno više vremena za tu operaciju. Takođe, nedostak je što jedan lekar može imati veliki broj zakazanih pregleda, dok neki drugi samo nekoliko, te neće svi serveri biti ravnomerno opterećeni.

- Treći pristup može biti vertikalno particionisanje. Prilikom dobavljanja termina ne trebaju nam informacije o korisniku poput "profesija", te ima smisla smisleno razdvojiti informacije.



Slika 2: Particije

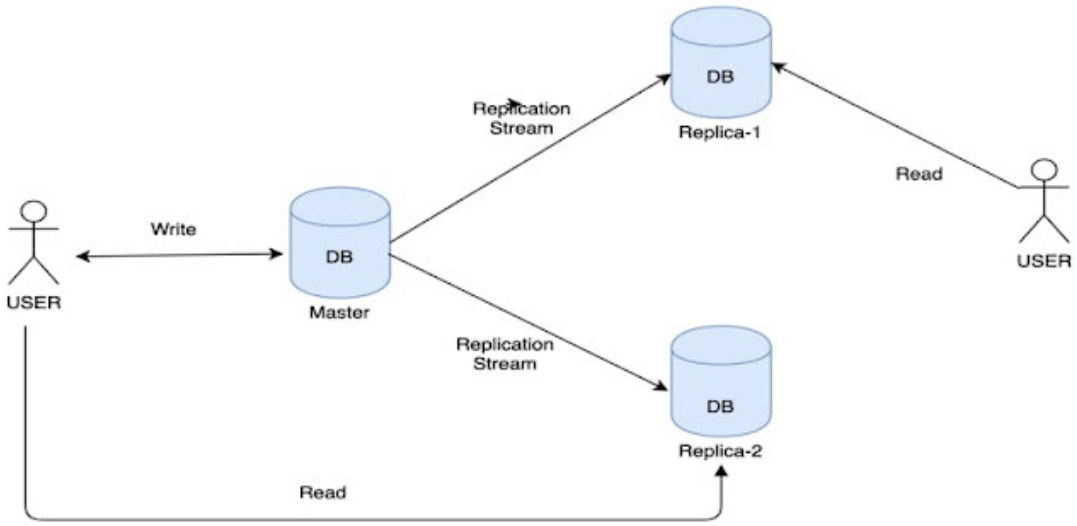
3 Predlog strategije za replikaciju baze i obezbeđivanje otpornosti na greške

Kopiranje podataka iz jedne baze u jednu ili više drugih obezbediće bolje performanse, pozudanost i veću dostupnost.

Da bi se izbegle situacije preopterećenosti baze velikim zahtevima i za pisanje i za čitanje, primenjuje se strategija za replikaciju baze gde imamo jednu primarnu (*master*) i nekoliko sekundarnih (*slave*) baza. Primarana baza služi isključivo da opsluži zahteve za pisanje, dok se čitanje vrši iz sekundarnih baza. Ovaj pristup omogućava da se podaci čitaju iz više različitih baza, što smanjuje opterećenje na glavnu bazu i povećava brzinu čitanja podataka. Takođe, ukoliko se ukaže potreba za pravljenje neke statistike ili izveštaja koji uključuje veliki broj podataka, za taj posao možemo angažovati *slave* baze, dok je primarna baza potpuno dostupna korisnicima.

Glavna prednost ove strategije replikacije je otpornost na otkaz. U slučaju da otkaže određena *slave* baza, neće nastati problem zato što u sistemu postoji nekoliko sekundarnih baza podataka. Ako pak *master* baza otkaže, jedna od *slave* baza će biti promovisana u *master*.

Što se tiče komunikacije između primarne i sekundarnih baza, biće korišćena asinhrona komunikacija. Ovakav vid komunikacije označava da *slave* baze podataka ne moraju da čekaju potvrdu od *master* baze da su podaci sinhronizovani pre nego što odgovore na zahtev za čitanjem podataka. Ovo može da poveća brzinu čitanja podataka, ali može dovesti do zastarelih podataka na *slave* bazama u slučaju da *master* baza ima visok stepen opterećenja. Zbog same namene i slučajeva korišćenja naše aplikacije, ovaj nedostatak se verovatno neće ispoljiti.



Slika 3: Master - Slave arhitektura

4 Hardverski resursi

Da bi se izračunala procena za hardverske resurse potrebne za skladištenje svih podataka u narednih 5 godina korišćene su sledeće pretpostavke:

- ukupan broj korisnika je veliki, preko 10 miliona,
- broj rezervacija svih entiteta na mesečnom nivou je 500.000,
- broj pregleda odgovara broju rezervacija na mesečnom nivou,
- sistem je skalabilan i visoko dostupan.

Potrebna memorija za čuvanje najčešćih entiteta u bazi (po torci):

- Address: 6KB,
- User: 24KB,
- Account: 16KB,
- Blood Bank: 6.4 KB,
- Qusetionnaire: 24KB,
- Appointment: 14KB.

Pretpostavimo da se broj pregleda u toku meseca poklapa sa brojem rezervacija (500.000). To znači da je na dnevnom nivou izvršeno $\frac{500.000}{30} \sim 17000$ pregleda. Ako prosečan član osoblja izvrši 10 pregleda u toku dana, sledi da nam je potrebno $\frac{17000}{10} \sim 1700$ zaposlenih na dužnosti po danu, pa je neki stvaran broj zaposlenih (bolovanje, godišnji odmori i zamosleni u administraciji) $5 \cdot 1700 \sim 10000$. U tom slučaju je broj korisnika $(10^6 - 10^4)$. Uzmimo da je 100 potreban broj centara za opsluživanje 17000 pregleda dnevno. U periodu o pet godina, broj termina iznosi $5 \cdot 12 \cdot 500000$. Tada proračun za zauzimanje memorije iznosi:

- Centri: $24KB \cdot 100 \sim 2400KB$
- Osoblje (informacije, nalog, adresa): $(24KB + 6KB + 16KB) \cdot 10000 \sim 0.44GB$
- Korisnici (informacije, nalog, adresa, upitnik):

$$(24KB + 6KB + 16KB + 24KB) \cdot (10^6 - 10^4) \sim 69.3 \cdot 10^6 KB = 66.09GB$$

- Termini: $14KB \cdot (5 \cdot 12 \cdot 500000) \sim 400.54GB$
- Slobodna procena za ostale entitete (oprema, uzorci krvi...): $5GB$

Ukupno zauzeće memorije za period od pet godina iznosi:

$$2400KB + 0.44GB + 66.09GB + 400.54GB + 5GB \sim 472.07GB$$

5 Predlog strategije za postavljanje load balansera

Radi ravnomernog prosleđivanja zahteva svim serverima, uvodi se load balanser.

U trenutnoj verziji projekta load balanser je implementiran upotrebom **Eureka Service Discovery** servisa. Eureka ima ugrađen load balanser koji zahteve prosleđuje po round-robin principu. Dobra strana je to što ovaj load balanser ima neka dodatna svojstva, poput automatskog detektovanja kad je određena instanca otkazala, i samim tim prestanak slanja zahteva ka toj instanci. Takođe, sve instance se periodično proveravaju da se potvrdi da su u dobrom stanju. Iako ovaj load balanser radi po vrlo jednostavnom principu, postoji opcija za unapređenje sa nekim malo naprednijim algoritmima poput *Weighted Round Robin*, koji nekim serverima prosleđuje više saobraćaja (korisno kada nisu svi serveri podjednakog kapaciteta i performansi), ili *Least connections*, koji prosleđuje zahtev ka onom serveru sa najmanje aktivnih konekcija i na taj način se teži ka tome da svi serveri budu podjednako efikasni.

U projektu je **API Gateway** integrisan sa Eurekom. API Gateway redirektuje zahteve ka servisu koji je Eureka pronašla kao slobodan. Dodatno, API Gateway podržava i druge tipove load-balansinga poput *sticky seasson*, koji prosleđuje sve zahteve od istog klijenta ka jednom serveru.

6 Predlog koje operacije korisnika treba nadgledati u cilju poboljšanja sistema

Modelovanje i praćenje promena ključnih entiteta kroz vreme nam doprinosi unapređivanju sistema. Opisani šablon jeste ***event sourcing***, i ovaj projekat sadrži veliki broj mesta gde se on može primeniti.

Rezervisanje termina kao glavni slučaj korišćenja ove aplikacije jeste nešto gde bi praćenje korisničkih operacija donelo najveće benefite. Informacije o korisniku koliko često ide da se pregleda ili da donira krv mogu poslužiti za kreiranje personalizovanih notifikacija radi podsećanja korisnika da je došlo vreme da ponovo ide na redovnu kontrolu, ili da donira krv. Takođe, prilikom rezervisanja možemo kvalitetnije predložiti neki od slobodnih termina ukoliko znamo navike korisnika, na primer da preferira termine pre podne ili kod određenog lekara. Pored unapređenja aplikacije, moguće je generalno unaprediti organizaciju. U slučaju da je određeni deo dana ili godine traženiji po pitanju termina, može se organizovati da u tom periodu više osoblja bude na dužnosti i samim tim postoji veći izbor slobodnih termina za korisnike.

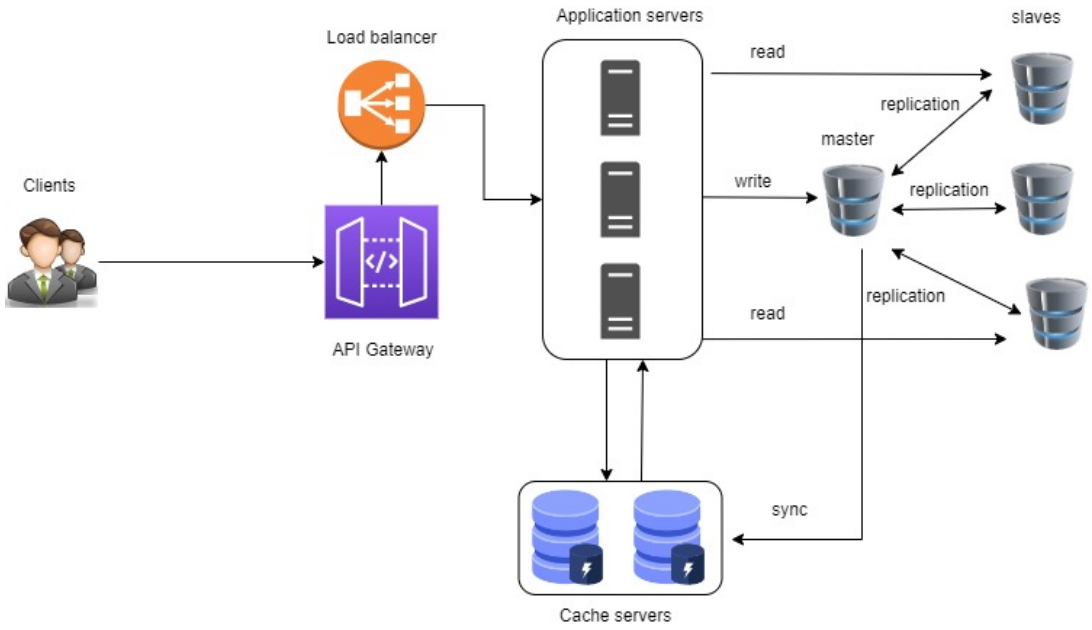
Dostava krvi bolnicama kao i razmena uzoraka krvi između centara je takođe zanimljiva za praćenje. Mogu se uočiti šabloni kojih tipova krvi je najviše poslato u određenu bolnicu/centar i na osnovu tih informacija bolje organizovati naredne isporuke. Dodatno, donacije krvi za krvne grupe koje su najtraženije mogu biti bolje promovisane.

7 Predlog strategije za keširanje podataka

Kako postoje podaci za koje postoji veći broj upita za čitanje nego za pisanje, ima smisla uvesti keširanje. U aplikaciji je podržan L1 nivo keširanja koji podrazumevano nidi Hibernet. Dodatno, nivo L2 je uveden uz pomoć bibliote EhChache. Na mikroprimeru su keširani lični podaci o korisniku i podaci o centru gde radi ulogovani administrator centra. Ove podatke ima smisla keširati zato što se retko menjaju. Takođe, bitno je sinhronizovati keširane podatke sa bazom podataka.

Kako bi u budućnosti aplikacija dostigla veliki broj korisnika, a samim tim i broj centara koji mogu biti rasporostranjeni širom sveta, ima smisla uvesti i *Content Delivery Network* (CDN) keširanje. CDN prestavlja mrežu servera na različitim geografskim lokacijama. Svaki zahtev od korisnika se šalje na njemu najblži server i na taj način se smanjuje vreme za dobavljanje sadržaja.

8 Kompletan crtež dizajna predložene arhitekture



Slika 4: Arhitektura sistema