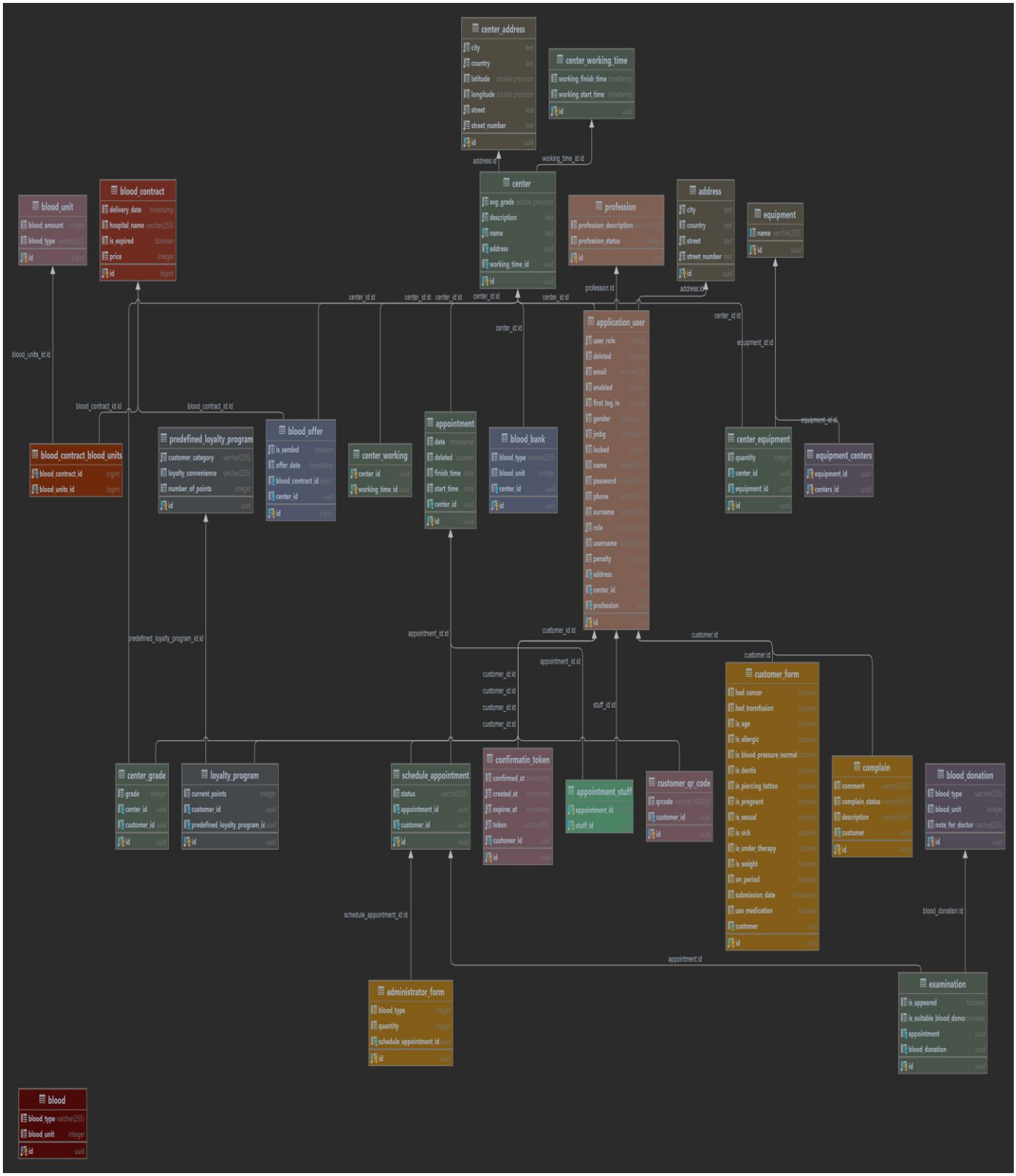
PROOF OF CONCEPT

SADRŽAJ

DIZAJN ŠEME BAZE PODATAKA

Dizajn šeme baze podataka predstavljen je pomoću relacionog dijagrama pomoću koga je ista i generisana (posredstvom *Spring Boot*-a). Za bazu podataka koriscen je PosgreSQL, a za ORM Hibernate.

Relacioni dijagram:



PREDLOG STRATEGIJE ZA PARTICIONISANJE PODATAKA

U velikim softverskim resenjima podaci se dele na particije kojima se nezavisno može pristupati i upravljati. Upotrebom particionisanja povećava se skalabilnost aplikacije i performanse.

Kada bismo pokušali sve podatke da skladištimo na jednom serveru vrlo brzo bi došli do hardverskog limita. Zbog toga se vrši sklabilnost i trudimo se naše podatke da rasporedimo na više servera. Jedan od najčešće korišćenih pristupa jeste **vertikalno i horizonalno particionisanje podataka**. Tabele koje imaju veliki broj entiteta možemo horizonalnim particionisanjem podijeliti na više manjih po nekom ključu. U slučaju naše aplikacije to bi bile tabele sa appointment-ima i scheduleAppointment-ima. Tako bi mogli da podelimo tabelu sa appointment-ima na osnovu centru kome priparada. Id centra se tako može proslediti hash funkciji koja će odrediti koji server baze podataka će čuvati appointment-e koje pripadaju određenom centru. Tabelu scheduleAppointment bi particionisali na osnovu Id-a donora. Takođe možemo izvršiti particionisanje appointment-a i scheduleAppointmenta i po vremenu njihovog nastanka. Appointment-i koje su se desili u prošlosti za nas nemaju veliki značaj već se koriste za neke vrste statistike. Za njihovo smestanje bi koristitili hardver lošijeg kvaliteta.

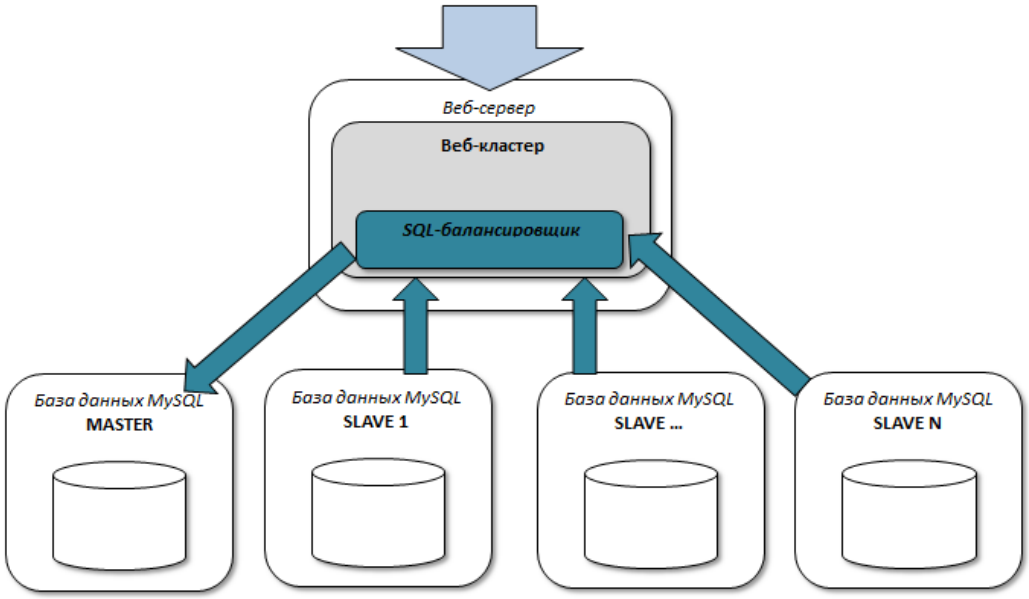
Particionisanje podataka takođe može u velikoj meri da poveća sigurnost naše aplikacije. Ideja bi bila da osetljive podatke čuvamo na posebnim serverima kod kojih bi primjenili veće sigurnosne mere. Tako možemo osetljive podatke korisnika(email, password, username) čuvati na posebnim serverima. Na taj način bi se i veličina upita smanjila jer se ovi podaci retko koriste osim pri autentifikaciji korisnika, dok drugim informacijama o korisnicima češće pristupamo. Potencijalno za cuvanje korisnika i njihovih kredencijala bi koristili third party library, kao na primer KeyClock api.

PREDLOG STRATEGIJE ZA REPLIKACIJU BAZE I OBEZBEĐIVANJE OTPORNOSTI NA GREŠKE

Jedna od tehnika koja se koristi da bi se povećala brzina aplikacije jeste da podielimo korisnike na osnovu njihovog geografskog položaja. Tako možemo postaviti servere koje su blizu svakoj grupi korisnika (na svakom kontintentu možemo staviti po jedan server). To će u velikoj meri povećati brzinu pristupa podacima. Ovi serveri bi bili međusobno povezani i radila bi se njihova sinhornizacija i na taj način bi korisnici u Evropi vidjeli iste podatke kao i oni u Americi.

Takođe bismo uveli i **replikaciju podataka**. Koristili bi **master i slave baze**. Podaci bi se nalazili na master bazi i na nju bi se vršilo pisanje i azuriranje podataka, dok bi se kopije nalazile na slave bazama. Slave baze repliciraju promene napravljene na glavnom serveru, sto omogucuje visestruku dostupnost istih podataka za koriscenje. Samo u master bazu je moguce upisivanje, a iz slave baza bi vrsili citanje. U sistemu bi se radila asinhrona sinhronizacija podataka između master i slave baza kako se ne bi mrežabila previse opterecena. Takođe ukoliko bi došli u situaciju da je master baza previše preoptrećna, zahtevi za čitanje podataka iz baze bi se slali ka slave bazama. Ukoliko bi došlo do otkaza master baze rad aplikacije se nastavlja korišćenjem slave baze. Na ovaj način je očuvana *otpornost na greške*. Isto tako ukoliko dođe do

neke prirodne katastrofe podaci će biti sačuvani jer imamo podatke na nasim slave bazama i pored toga na svakom kontintentu gde se naša aplikacija koristi imamo servere sa bazama podataka.



PREDLOG STRATEGIJE ZA KEŠIRANJE PODATAKA

Keširanje podataka je nezaobolazna stavka prilikom visoko rastućih sistema. Velika količina podatka, kao i poziva na bazu mogu znatno da uspore sistem, a time i učine korisničko iskutvo manje prijatnim, odbijajući klijente od korišćenja veb aplikacije.

Ovaj proces najbolje je implementirati za **keširanje primarno statičkih podataka**, kao što su podaci o centrima, opremi, slike,qr-kodovi. Ukoliko pretpostavimo da je na mesečnom nivou broj appointment-a milion ili vise, možemo videti da su podaci visoko promenljivi. Često će se zakazivati novi appointment-i, te će se time dostupnosti entiteta rapidno menjati.

Ipak, i ove podatke možemo sačuvati u keš memoriji, s time što bismo ih menjali u kešu svaki put kada se oni ažuriraju u bazi.

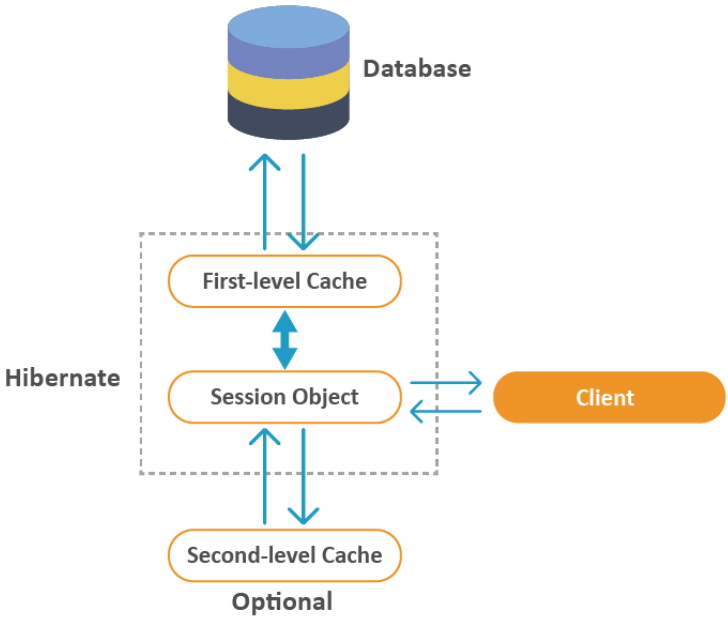
Ovaj proces možemo da optimizujemo uklapajući ga sa **Event Sourcing**-om, opisanom u nastavku dokumenta. Za ograničenu keš memoriju, najpovoljnije je keširati podatke za koje znamo da se često dobavljaju: u nasem slucaju rad sa appointment-ima. Pomocu Event Sourcing-a mozemo zabeleziti koliko cesto i u kom periodi se appointment-i zakazuju, na taj nacin mozemo izvuci neke statistike pomocu kojih mozemo odrediti koliko unapred korisnici zakazuju iste.

Ukoliko se keš napuni, dobra strategija je ***LRU – Last Recently Used***, kojom se izbacuju najstariji dobavljeni podaci.

Takodje koristili bi TTL (Time To Live) strategiju gde bi za podatke koje se frekventije menjaju stavili da se kes podaci cesce azuriraju.

CDN (Content Delivery Network) keširanje bi takođe bilo povoljno implementirati, kako će ono pomoći prilikom bržeg dobavljanja statičkih podataka, uključujući slike, i veb stranice. Kako je posrednik između podataka i klijenta u ovom slušaju proxy server lociran korisniku bliže u odnosu na originalan server aplikacije, očekivano je da će i traženi podaci stići brže. Ova implementacija posebno je korisna ukoliko uvedemo pretpostavku da se shodno velikom broju korisnika (100 miliona) naša aplikacija koristi internacionalno.

Aplikacija **BloodBank** trenutno implementina na globalnom nivou najznacajnijih entiteta kao sto su lista centara, QR kodova i appointment-i ciji je TTL postavljen na 10s. Korišćen je **EnCache**, jer je pouzdana biblioteka za kesiranje podataka i jednostavna sa Spring Boot aplikacije.



OKVIRNA PROCENA ZA HARDVERSKE RESURSE POTREBNE ZA SKLADIŠTENJE SVIH PODATAKA U NAREDNIH 5 GODINA

Pretpostavke sistema:

- Ukupan broj korisnika: 10 MILIONA
- Broj rezervacija svih entiteta na mesečnom nivou: 500000
- Sistem mora biti SKALABILAN I VISOKO DOSTUPAN

Zauzetost memorije [1] za skladištenje 10 najvecih entiteta u bazi[2]:

	relation	total_size
1	center	48 kB
2	equipment	40 kB
3	blood_donation	32 kB
4	customer_qr_code	32 kB
5	center_address	32 kB
6	address	32 kB
7	predefined_loyalty_program	32 kB
8	application_user	32 kB
9	appointment	24 kB
10	blood	24 kB

Za 10 miliona korisnika potrebna memorija iznosi: **280 GB**.
Za procenjenih 20 000 000 entiteta potrebno je **2900 GB**.
Za pola miliona appointment-a mesečno, tokom godinu dana potrebno je **20 GB** za scheduleAppointmnt-e **10 GB** za brze rezervacije, ukupno **30 GB**.
Za skladištenje drugih entiteta slobodnom procenom izdvojićemo **30 GB**.

Napomena: Sve procene rađene su za situacije koje bi zauzele više memorije, kako bi se izbeglo podcenjivanje potrebnih resursa.

UKUPNO, ZA ODRŽAVANJE SISTEMA U TRAJANJU OD 5 GODINA POTREBNO JE 16050GB.

PREDLOG STRATEGIJE ZA POSTAVLJANJE LOAD BALANSERA

Ideja upotrebe load balancera jeste da opterećenje prenesemo na više servera i da jedan server ne obrađuje sve zahtjeve koje mu šalju klijenti. Stoga ćemo postaviti load balancer između klijenta i aplikativnog servera.

Prema nekim istraživanjima algoritam za load balancing koji daje dobre rezultate jeste „**Least Pending Requests**“. Ideja algoritma jeste da se prati broj zahtevaza odredjeni servera bi se dijagnostikovalo da li neki od servera preopterecen. U realnom vremenu se prati broj zahteva koji svaki server treba da obradi i na najmanje opterecen server se salje zahtev. Algoritam može efikasno da se nosi sa kašnjenjem servera i

[1] PostgreSQL Table Size

vremenskim ograničenjima za zahteve. On automatski prepoznaje sve činioce koji će dovesti do produžetka redova zahteva te nove zahteve prosleđuje na manje opterećene servere.

Mi smo u nasoj aplikaciji na globalnom nivou postavili Eureka Netflix load balancer, tako sto smo napravili jos jednu Spring Boot aplikaciju koja predstavlja server koji vrsi load balancing za hospital aplikaciju i nasu BloodBank aplikaciju koje smo registrovali na taj server.

PREDLOG KOJE OPERACIJE KORISNIKA TREBA NADGLEDATI U CILJU POBOLJŠANJA SISTEMA

U cilju kreacije kvalitetne aplikacije glavni prioritet ne bi trebao da bude samo pravljenje tehnički ispravne i optimizovane aplikacije, već i one koja će svojim ponašanjem da reflektuje personalne potrebe svojih korisnika. U tu svrhu je **Event Sourcing** jedan od principa koje je potrebno primeniti na što širem spektru.

Pre svega, sistem **pretrage** i **pregleda stranica** posecenih centara, zakazanih appointment-a bio bi značajan pokazatelj zainteresovanosti klijenta. Vreme provedeno na svakoj od pojedinih stranica znatno doprinosu stvaranju realne slike koliko korisnici zainteresovani za iste.

Ns taj nain aplikacija bi na osnovu ovih rezultata bila lako proširiva pokazivanjem „*Viewed most often*“ ili „*Recommendations*“ prikaza kako registrovanim donorima, tako i neautentifikovanim korisnicima. Dodatno, admini ili administratori centra mogli bi da imaju pristup grafičkom prikazu gde bi na osnovu datih informacija priložio pravu sliku o njihovim najuspešnijim i najtraženijim centrima, na osnovu kojih bi mogli poboljsali ostale usluge.

Dodatna akcija koja se prati mogla bi da bude prilikom samog procesa zakazivanja appointmenta donora. Koliko često donori dođu do stranice pravljenja rezervacije i odustanu od iste? Koliko često korisnici otkažu prethodno zakazan appointment? Koliko cesto korisnici doniraju krv?

Sledeca akcija bi bila koliko koraka je potrebno medicinskom osoblju da izvrse zakazan pregled, na kojim koracima provodi najvise vremena, kao i koliko vremena prevodi na svakom koraku.

Dodatno napravili bi servis koji bi na osnovu algoritma masinskog ucenja predlagao vreme i centar u kome je donoru preporuceno da rezervise appointment.

Za sistem je veoma bitan tok informacija između njenih korisnika. Primarno, ocene koje klijenti ostavljaju čine sistem bogatijim podacima, i lakše koristivim za nove korisnike. **Praćenje logovanja** na sitemu, kao i **ostavljanja komentara** za zavrsene appointment-e feature u aplikaciji, time bi omogucili da svaki centar poboljsa rad na osnovu iskustva donora. Na primer, ukoliko je vise donaora imalo lose iskustvo u nekom centru , tom centru se salje email kao vrsta upozorenja da nesto nije u redu sa tim centrom.

KOMPLETAN CRTEŽ DIZAJNA PREDLOŽENE ARHITEKTURE