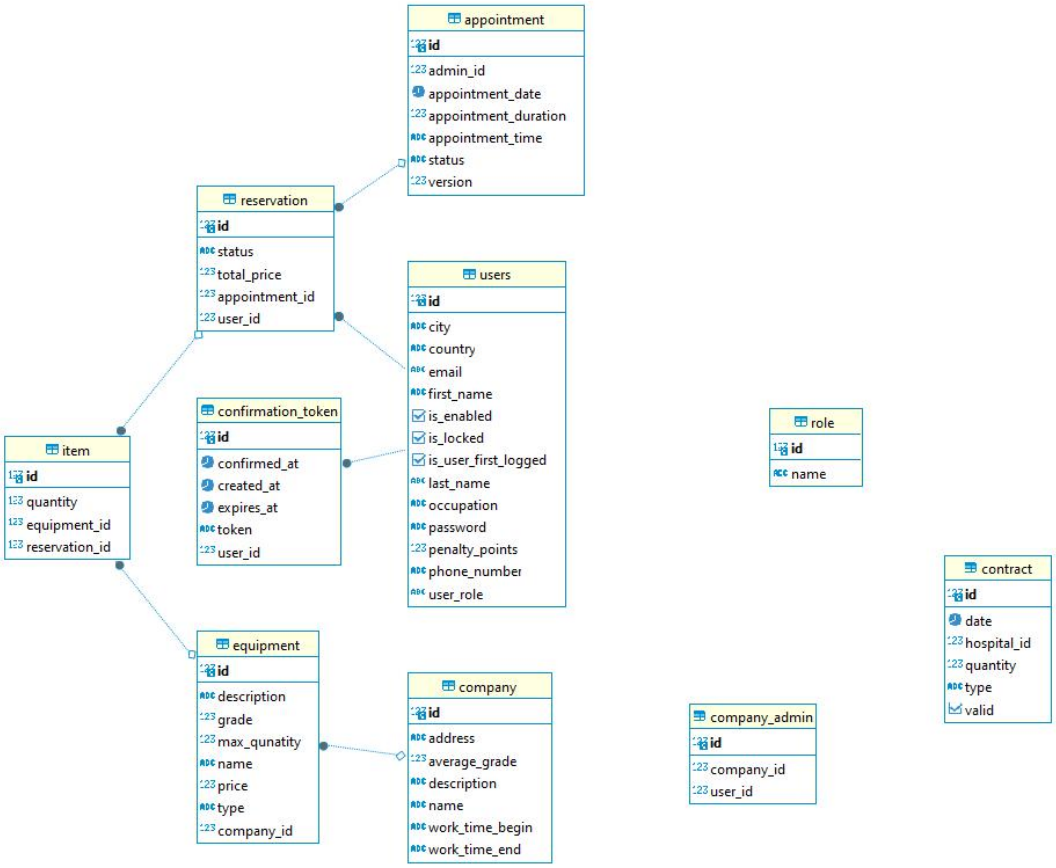
Predlog arhitekture aplikacije

Ovaj dokument predstavlja predlog koncepta kako bi trebalo da izgleda arhitektura sistema za upravljanje medicinskom opremom kada bi broj istovremenih korisnika prerastao mogućnosti jednog servera. Predložena arhitektura je koncipirana tako da zadovoljava zahteve skalabilnosti i visoke dostupnosti. Pretpostavka je da je broj korisnika aplikacije preko 100 miliona, kao i da broj rezervacija koje korisnici izvrše na mesešnom nivou prelazi 500 hiljada. Kroz sledećih nekoliko tačaka, biće predstavljena neka od mogućih rešenja za dizajn arhitekture koja bi podržala sve navedene pretpostavke.

Dizajn šeme baze podataka

Na slici ispod, prikazana je logička šema baze podataka sistema.



Slika 1 – Šema baze podataka

Predlog strategije particionisanja podataka

Particionisanje podataka može biti izvršeno na dva načina, vertikalno ili horizontalno, gde god se primeti da je češća upotrebi nekih podataka unutar jedne tabele u odnosu na druge.

S obzriom na činjenicu da je baza podataka normalizovana, nema prevelike potrebe za dodatnim vertikalnim particionisanjem tabela, ali se može postići ubrzanje korišćenjem Lazy Loading mehanizma u Objektno-Relacionom maperu. Ovo se postiže uz pomoć anotacije **fetchtype.LAZY**, gde se, na primer, prilikom dobavljanja podataka o kompanijama, podaci o njihovim adminima dobavljaju samo onda kada se eksplicitno zatraže.

Prilikom razmatranja o upotrebi horizontalnog particionisanja, primećeno je da je moguće podeliti tabelu termina po mesecima na datuma zakazivanja, jer se može uzeti pretpostavka da će se svakog meseca desiti ravnomeran broj termina. Ovim se olakšava pretraživanje termina koji su bili izvršeni u prethodnim periodima.

Predlog strategije za replikaciju baze podataka i obezbeđivanje otpornosti na greške

Generalno, najveći problem trenutne arhitekture je taj što baza podataka nije replicirana, pa u slučaju otkazivanja instance baze, sistem prestaje sa radom. Jedno od najkorisćenijih rešenja za ovaj problem je upotreba **master-slave arhitekture**. Master instanca baze podataka zadužena je za upisivanje, a po potrebi i za čitanje, a slave instance samo za čitanje podataka. Takodje, master instanca ima ključnu ulogu u održavanju konzistentnosti podataka svih ostalih instanci baze. Ovaj proces se može obavljati sinhrono ili asinhrono, u zavisnosti od potreba i namena informacionog sistema.S obzirom na to da su podaci o dostupnoj opremi i zakazanim terminima od izuzetnog značaja za korisnike, od velike je važnosti da u svakom trenutku sve instance baze imaju konzistentne podatke, pa bi logičan izbor bila sinhrona arhitektura.

Predlog strategije za keširanje podataka

Keširanje u treuntnom sistemu je podržano na više nivoa. L1 cache je podržan od strane Hibernate objektno-relacionog mapera, a za eksterni, odnosno L2 cache koristi se **EhCache**. Za strategiju keširanja odabrana je **read_write** strategija, a što se tiče odabira podataka koji se keširaju, razmatraju se podaci koji se frekventno koriste i retko menjaju. Za potrebe sistema to bi bili podaci o kompanijama.

Okvirna procena hardverskih resursa potrebnih za skladištenje svih podataka u narednih 5 godina

Kroz narednih pet godina, očekuje se 100 miliona korisnika sa brojem izvršenih rezervacija koji je preko 500 hiljada na mesečnom nivou, stoga najveću količinu memorije će uzeti entiteti korisnik i rezervacija, dok će ostali entiteti biti nekoliko redova veličina u odnosu na broj korisnika i rezervacija.

Zbog prethodno navedenih činjenica, ovaj odeljak će se fokusirati na procenu potrebnih resursa za skladištenje korisnika i rezrevacija, pri čemu se očekuje da korisnika bude 100 miliona a rezervacija 30 miliona. Pošto je entitet termin povezan sa rezervacijama, on će takođe biti uračunat u ovu procenu.

String tip podataka u javi je, u osnovi, UTF-16 enkodiran, što znači da je jedan karakter veličine 2 bajta. Zato način preračunavanja veličine String podataka će biti izveden tako što se maksimalna dozvoljena veličina String-a pomnoži sa dva. Takođe, String će biti uvršten u primitivne tipove, kako se ne bi stalno posebno naglašavao. Ono što je još bitno napomenuti jeste da se boolean tip ne uzima u obzir u razmatranju jer zauzima samo jedan bit.

Entitet korisnik se sastoji od 9 String podataka maksimalne dužine 255 karaktera, i od po jednog integer i double podatka, što je ukupno 4602 bajta. Ako se uzme pretpostavka da će sistem imati 100 miliona korisnika, to dovodi do cifre od 428 terabajta podataka. Ovo bi se moglo umanjiti ukoliko bi se String tipovi ograničili na razumnije dužine karaktera.

Pošto entitet rezervacija u sebi sadrži reference na korisnika i termin, kako bi se razmotrili hardverski resursi neophodni za skladištenje podataka o rezervacija, treba uzeti u obzir veličinu jednog sloga entiteta termin. Entitet termin sadrži u sebi 3 String podatka dužine 255 karaktera i jedan podatak tipa integer. Uzimajući u obzir da će za pet godina u sistemu biti 30 miliona rezervacija, odnosno termina, ovo iziskuje potrebu za dodatnih 46 terabajta skladišnog prostora.

Sveukupno, procena za narednih pet godina je otprilike oko 475 terabajta skladišnog prostora.

Predlog strategije za postavljanje load balancera

Što se tiče strategije za postavljanje load balancera, adekvatna strategija za sistem bi bila „Weighted Round Robin“. Ovakva strategija bi nam omogućila da saobraćaj raspodelimo u zavisnosti od jačine raspoloživih servera. U slučaju da je potrebno opsluživati korisnike na različitim geografskim lokacijama, ova strageija nam omogućava da saobraćaj rasporedimo i po tom parametru.

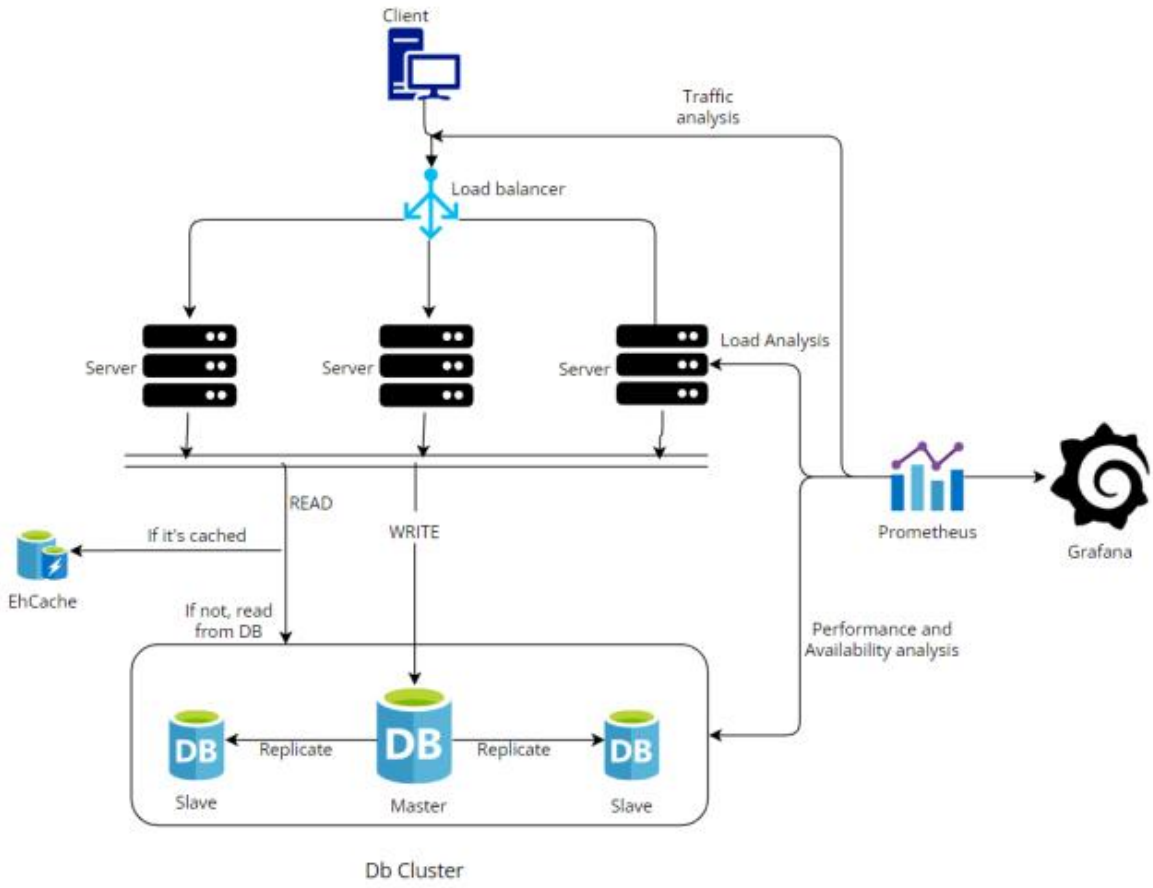
Predlog operacija za nadgledanje

Nadgledanje rada sistema je od ključne važnosti za donošenje odluka o sklairanju neophodnog broja servera. Praćenjem prometa u različitim vremenskim periodima nam može ukazati na to da li je potrebno povećati broj servera zbog povećanog saobraćaja u sistemu, kao i da li u nekom periodu radi nepotrebno veliki broj servera. Sve ovo je realizovano korišćenjem Prometheus alata za prikupljanje i skladištenje metrika o aplikaciji, pomoću kojeg je moguće meriti performanse servera i baze podataka. Kako bi se adekvatno ovi podaci analizirali, koristi se i Grafana kao alat za vizuelizaciju trenutnog stanja aplikacije.

Takođe, još jedan od glavnih razloga za praćenje rada aplikacije jeste spoznavanje potreba korisnika sa ciljem poboljšanja korisničkog iskustva, što se može postići mehanizmom event sourcing-a.

Predlog arhitekture

Na narednoj slici prikazan je predlog arhitekture:



Slika 2: Predlog arhitekture sistema