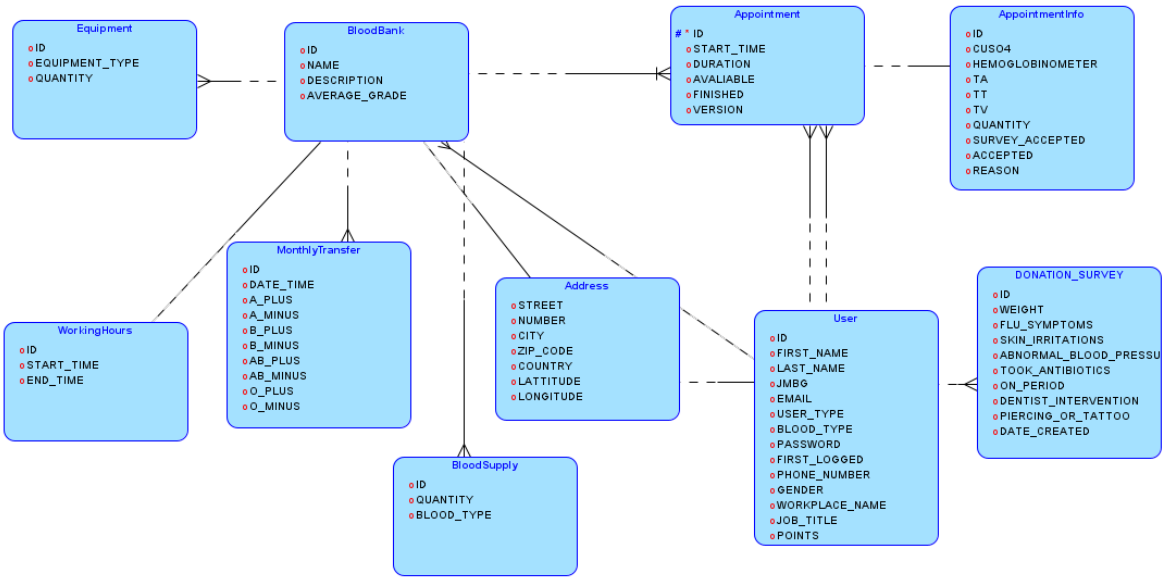
# Dizajn šeme baze podataka




# Predlog strategije za particionisanje podataka

Predlažemo particionisanje podataka po učestalosti čitanja tj. obezbedili bismo da se podaci po tabelama particionišu tako da se podele na one koji se često čitaju i na one koji se retko čitaju. Konkretno, primer za ovo bi bila tabela “Address” čiji se podaci koriste samo prilikom pregleda informacija o korisniku ili banke krvi. Takođe postoje podaci koji su dodatno mogli da se izdvoje na primer tabela “User” koristi polja “firstLogged”, “phoneNumber” “workplaceName”, “jobTitle” koji se veoma retko koriste pa bi se takvi podaci mogli prebaciti u zasebnu tabelu sa nazivom “UserDetails”. Još jedan primer dobrog particionisanja podataka jeste tabela “WorkingHours” čiji podaci se čitaju samo prilikom zakazivanja appointment-a.

# Predlog strategije za replikaciju baze i obezbeđivanje otpornosti na greške

Strategija za replikaciju baze i obezbeđivanja otpornosti na greške u našem slučaju podrazumevala bi kreiranje master-slave konfiguraciju više povezanih baza podataka.Master baza(ponekad nazvana i publication ili primary) baza bi podržavala samo operacije upisa(INSERT, DELETE, UPDATE). Pored master baze postojalo bi i više slave(secondary,

subscription) baza koje bi bile read-only dakle one bi se koristile samo za čitanje podataka jer je to operacija koja se najčešće izvršava. Tokom rada na projektu koristili smo MS Azure SQL database i SQL database server kako bismo hostovali našu bazu na cloudu. Ovakvo rešenje bi bilo relativno lako unaprediti dodavanjem više instanci i zakazivanjem redovnih backup-ova ali takvo rešenje podrazumeva veliko trošenje resursa.

# Predlog strategije za keširanje podataka

Pošto spring boot već podržava keširanje podatak uz pomoć svog cache dependency-ja, implementacija ovakvog sistema bi bila prilično jednostavna.

Što se tiče in-memory caching-a koristili bismo Ehcache zbog svoje popularnosti, mogućnosti čuvanja ogromnog broja informacija, brzog fetch-ovanja podataka kao i odlične horizontalne skalabilnosti.

Zbog velikog broja statičkih podataka na našoj web stranici CDN je neophodan ali srećom po nas on je u velikoj meri integrisan sa modernim browserima tako da nije potreban nikakav setup.

Kesiranje je u aplikaciji implementirano nad pretragom korisnika i podacima o appointmentima pomocu nonrestricted read write strategije koja je trenutno dovoljna da zadovolji uslove obzirom da nam sistem nije visoko konkurentan i da eventualno zastereli podaci nisu kritican problem. Pored navedenih funkcionalnosti, kesiranje bismo mogli implementirati nad jos nekim funkcionalnostima, kao sto su npr podaci o bankama krvi.

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

Za procenu veličine samih podataka uzimali smo vrednosti veličine primitvnihj tipova podataka u javi kao i aproksimaciju da je jedan java.String na 64bitnom sistemu zauzima 32 + string.length() * 2 došli smo do zaključka da će za 5 godina biti potrebno:

Za korisnike: 613B + 222B(za adresu) = 835 B * 10 miliona korisnika = 7.7 GB

Za appointment: 192B * 500k * 12 meseci * 5 godina = 5.36 GB

Za ostale entitete u sistemu: 3.8GB

Ukupno **16.86 GB**

# Predlog strategije za postavljanje load balansera

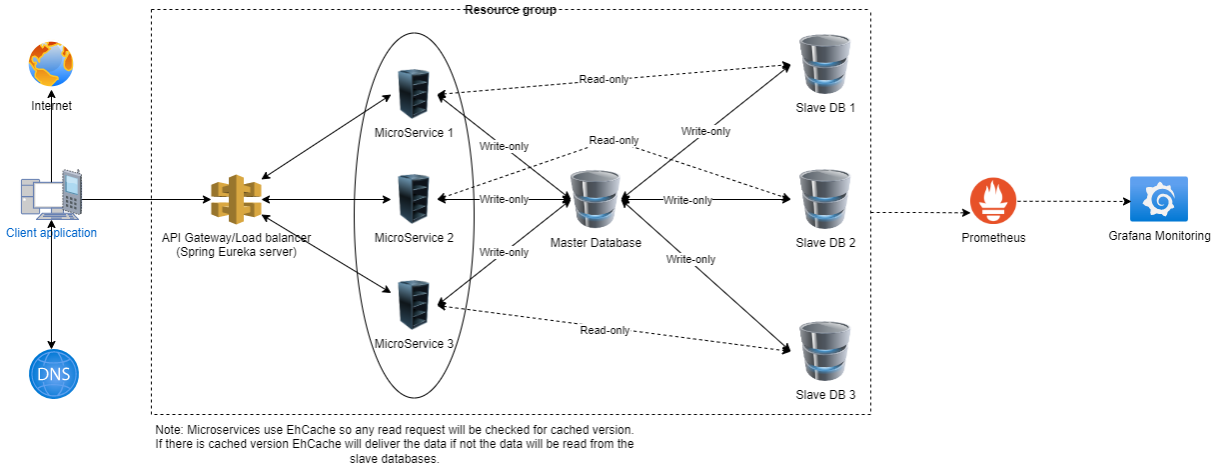
U cilju maksimalnog pojednostavljenja našeg rešenja kao i smanjenjivanja kompleknosti samog sistema opredelili smo se za već napravljena rešenja koja se mogu iskoristiti na cloud servisima poput Amazon Web Services(Elastic Load Balancing) i Microsoft Azure(Azure Load Balancer) koji koriste Round Robin i Least Outstanding Requests algoritam za prosleđivanje zahteva ka serverima.

# Predlog koje operacije korisnika treba nadgledati u cilju poboljšanja sistema

Za nadgledanje korišćenja aplikacije, monitoring, u okviru projekta koristili smo Spring Boot Actuator, u saradnji sa Prometheus-om za prikupljanje informacija i Grafana-u grafički prikaz. To je omogućilo kvalitetan pregled korišćenja aplikacije, broja konekcija, trajanja pristupa i velikog broja drugih informacija. Za monitoring posebno su važne informacije kao što su response time, resource usage, application availability ili error rates.

U slučaju konkretne implementacije praćeno je korišćenje cele aplikacije, ali slučajevi na koje bi u realnoj situcaiji trebalo obratiti pažnju su zakazivanje i kreiranje termina gde se korisniku predlažu neki termini i praćenje response time, kao i povratna informacija za skeniranje qr koda koje bi trebalo da bude što brže. Takođe bilo bi korisno ispratiti i praćenje istorije pretrag da bi se korisniku pružio željeni sadržaj.

# Dizajn predložene arhitekture




Note: Microservices use EhCache so any read request will be checked for cached version. If there is cached version EhCache will deliver the data if not the data will be read from the slave databases.