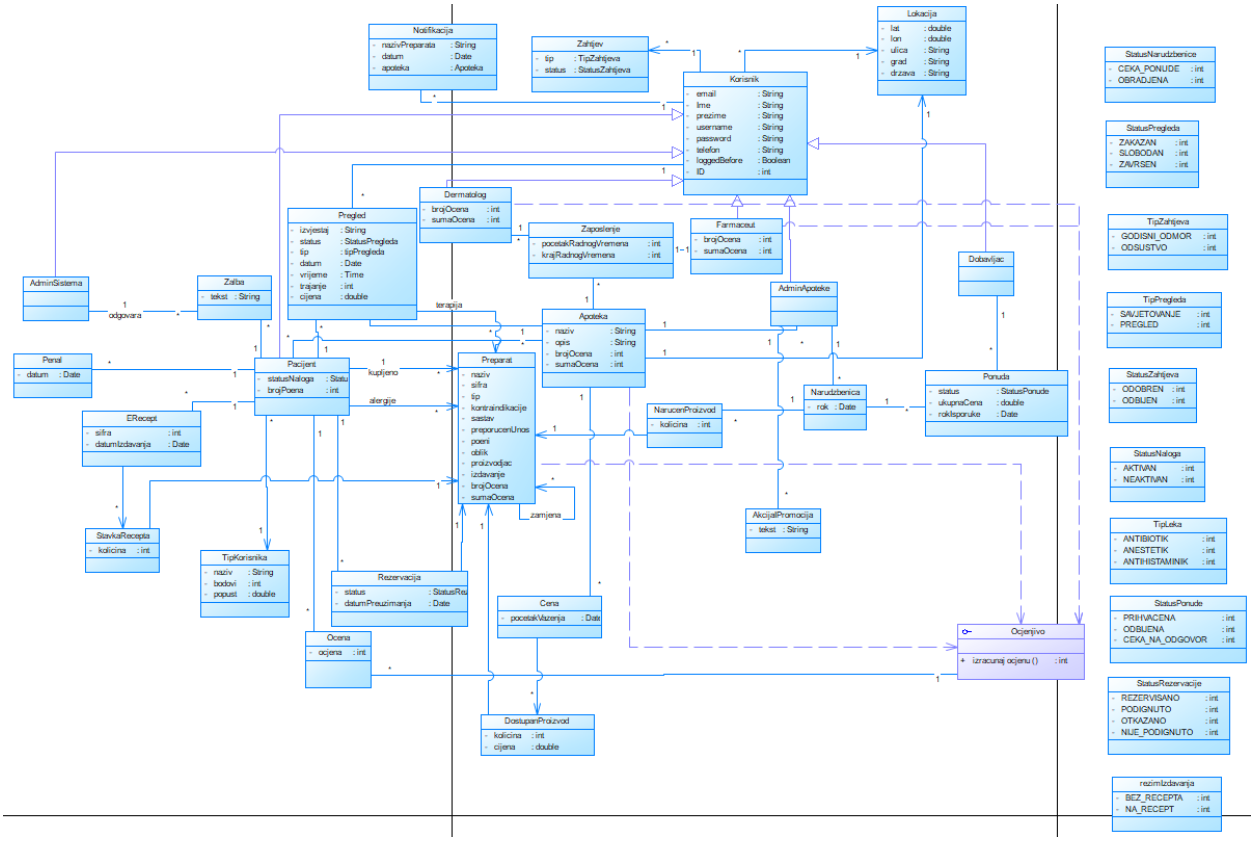
Proof Of Concept

ISA_2021-tim6

1. Arhitektura web aplikacije (dijagram klasa):



2. Strategija particionisanja podataka:

Generalno, za svaku tabelu u bazi podataka gdje postoje podaci koji se cesto ne citaju mozemo izvrsiti particionisanje. Jednostavan rule-of-thumb bi bio da svuda gdje je učitavanje u kodu anotirano sa *LazyLoading* anotacijom tabelu možemo podeliti na dve: jedna koja sadrži podatke koji su logični da su korisni prilikom svakog čitanja i onu koja sadrži dodatne informacije. Na primer, kod tabele Korisnici, opšte informacije o svakom korisniku kao i informacije o pregledima i alergijama bi mogle biti u jednoj tabeli dok su podaci o akcijama, promocijama, rezervacijama i žalbama u drugoj. Takođe, geografsko particionisanje uvek možemo primeniti bez obzira na arhitekturu (s druge strane, koliko je ovo neophodno u našoj aplikaciji je diskutabilno).

Horizontalno particionisanje bi u našoj aplikaciji bilo od velike koristi. Preglede, savetovanja, rezervacije i e-recepte bi mogli particionisati u tabele koje bi predstavljale mesece u godini (Januar, Februar, ... , Decembar) i kvartale (Prvi, ..., Četvrti) te bi se ispis izveštaja ili radnog kalendara učinio efikasnijim.

Ideju za dalje particionisanje (hipotetički) možemo dobiti u budućnosti korišćenjem metoda iz poglavlja [7] ovog dokumenta.

3. Strategija za replikaciju baze i obezbeđivanje otpornosti na greške:

Trenutno najveći problem koji naša BP ima je postojanje *single point of failure*-a, odnosno, pošto imamo samo jednu instancu baze, njeno otkazivanje bi značilo pad čitavog sistema. Ovo možemo rešiti tako što ćemo uvesti još nekoliko redundantnih instanca baze podataka i uvezati ih u standardnu *master-slave* (*primary-secondary*) konfiguraciju. Ovim će svi *read-only* zahevi stizati na *slave* instance te će se tako rasteretiti *master*. *Write* zaheve obrađuje isključivo *master* i on je zadužen za očuvanje konzistentnosti u sistemu. Način očuvanja konzistentnosti (tj. propagacije promena) bi mogla biti jednostavna propagacija operacija koje su dovele do promene, ili pak propagacija samih podataka (malo sporije). Kao dodatan mehanizam otpornosti na greške, od *slave* instanci se može tražiti povratna informacija da li je ažuriranje uspešno izvršeno nakon čega se može ponoviti operacija propagacije. I master i slave DB bi mogle biti instance naše već postojeće *Postgresql* baze s obzirom da je ista relaciona i samim tim strukturirana i laka za održavanje. Što se tiče alata potrebnih za replikaciju možemo koristiti *Veeam Backup & Replication* s obzirom da se vodi kao jedan od najpouzadanijih alata za ovaj problemski domen i što je najbitnije, podržava rad sa *Postgresql* bazom i ima ugrađen *failover* mehanizam. *Veeam* koristi *image-based* VM replikaciju.

4. Strategija za keširanje podataka:

Kako je *L1 cache* podržan po *default*-u od strane *Hibernate*-a isti je već uključen u našu aplikaciju. Što se tiče *L2 cache*-a, *Hibernate* podržava i ovaj način keširanja, međutim potreban je neki eksterni provajder kako bi se isti podesio. Naš izbor bi bio *EhCache* s obzirom da podržava sve strategije keširanja a i predstavlja jedan od najboljih i najpopularnijih provajdera za *L2* keširanje u Spring aplikacijama. Ideja je da se podaci koji se retko ažuriraju keširaju i tako se optimizuje pristup istima. U slučaju naše aplikacije to bi bili podaci o apotekama i preparatima u sistemu. Konkretna strategija keširanja bi bila *Transactional*. Ovo može izgledati kao svojevrsni "*overkill*" s obzirom da se *transactional* strategija mahom koristi u visoko konkurentnim sistemima i da bi *nonrestricted read-write* verovatno bio dovoljan da se zadovolje uslovi, međutim, *transactional* nam obezbeđuje da će se baza podataka zaštititi da se u istoj nađu zastareli podaci koji neće adekvatno oslikavati realni sistem.

Sa druge strane bitno je omogućiti i CDN keširanje koje će da učine statičke podatke bliže korisniku te da se oni brže učitaju kod korisnika, pogotovo zbog velikog broja *.js* fajlova koje sadrži naša aplikacija. Sreća kod ovoga je da veliki broj modernih *web browser*-a podržavaju ovaj vid keširanja tako da nije potrebno uključivati neki dodatan servis.

5. Okvirna procena hardverskih resursa potrebnih za skladištenje svih podataka (za narednih 5 godina):

1.1. Skladištenje pacijenata:

S obzirom da podaci o jednom pacijentu zauzimaju oko 1618 bajtova ili 1.6 KB, podaci za 200 miliona korisnika će iznositi priblizno 305 GB.

1.2. Skladištenje rezervacija preparata:

S obzirom da podaci o jednoj rezervaciji zauzimaju oko 40 bajta, a da je procenjena mesečna kvota oko milion rezervacija, za period od 5 godina dolazimo do brojke od približno 2.2 GB.

1.3. Skladištenje izveštaja o pregledima/savetovanjima:

S obzirom da podaci o jednom pregledu/savetovanju zauzimaju 1056 bajta ili približno 1.03 KB, a da je procenjena mesečna kvota oko milion pregleda/savetovanja, za period od 5 godina dolazimo do brojke od približno 59 GB.

1.4. Skladištenje preparata

S obzirom da podaci o pojedinačnom preparatu zauzimaju oko 1.28 KB, pod pretpostavkom da ćemo za 5 godina imati oko milion razlicitih preparata u sistemu, dolazimo do brojke od 1.22 GB.

6. Strategija za postavljanje load balansera:

S obzirom da naša aplikacija ne implementira REST u potpunosti već se autentifikacija obavlja preko sesije, horizontalno skaliranje je nešto složenije nego da imamo potpunu *stateless* komunikaciju (gdje bi smo samo ukljucili vise masina u sistem). Naš *load balanser* bi u tom slučaju morao da poseduje mehanizam *sticky session*-a gde bi zapamtio koji korisnik je ulogovan na koji server i njegove zahteve prosleđivao samo tom serveru. Ogroman problem ovde bi predstavljalo otkazivanje nekog od servera. Rešenje za to bi bilo korišćenje *Infinispan* baze podataka koja bi bila instalirana na svakom serveru u *cluster*-u i koja bi se mogla ugraditi u *SpringBoot* aplikaciju kao obična biblioteka, te bi podatke vezane za autentifikaciju vezali za nju i imali bi pristup istima bez obzira na kom serveru se obrađuje zahtev. Mana ovog pristupa je velika cena *Infinispan* baze. Sa druge strane, *Redis* baza je podržana *out-of-the-box* u *Spring*-u i moze nam pružiti besplatno rešenje za ovaj problem, na mali uštrb brzine pristupa što i ne bi bio problem s obzirom da podaci potrebni za autentifikaciju sadrže samo najosnovnije podatke o korisniku te bi *Infinispan* optimizacije za pristup velikim objektima ovde bile suvišne.

Još jedan moguć pristup ovde bi bio prelazak na autentifikaciju koriscenjem JWT-a. Time bi omogućili da čak i jednostavan *nginx*-ov, *round robin* load balancer može ispuniti naše zahtjeve za horizontalnim skaliranjem bez uvođenja dodatnih mehanizama za prostiranje sesije između različitih node-ova u cluster-u.

Što se vertikalnog skaliranja tiče, to uvek možemo primeniti bez obzira na arhitekturu sistema tako što ćemo na sve mašine staviti više memorije (standard za servere je 3GB, međutim, razumno je ići i do 64GB za veće aplikacije) i jači CPU (trenutno najbolji CPU za *high-end workstation* sisteme je AMD Threadripper 3995WX).

7. Predlog operacija za nadgledanje:

Smatramo da je praćenje ponašanja korisnika ključno u poboljšanju rada našeg web servisa. Dakle, sve informacije o korisnicima (lične informacije i informacije o alergijama) kao i rezervacijama (bilo ličnih korisnikovih rezervacija ili onih koje su mu dermatolog/farmaceut napravili) i zakazivanjima (pregleda i savetovanja) kao i e-receptima su veoma važne da se nadgledaju. Kao dodatnu funkcionalnost moguće je ubaciti mehanizam praćenja istorije korisnikovih pretraga (apoteka, preparata i zaposlenih) koje bi se naknadno mogle obraditi i klasterovati i kao takve upotrebiti u pravljenju nenadgledanog ML modela koji bi korisniku mogao da preporučuje dodatan sadržaj koji ga interesuje (reklame) kao i da pomogne nama

da steknemo uvid u navike korisnika kako bi mogli poboljšati postojeće ili uvesti nove funkcionalnosti, kao i možda da kreiramo novu taktiku particionisanja BP kako bi rasteretili sistem.

8. Predlog arhitekture:

Predstavljena je arhitektura ako se uzme u obzir trenutna verzija autentifikacije (korišćenjem sesije), ukoliko bi prešli na JWT autentifikaciju, arhitektura bi ostala ista izuzev *Redis* baze koja ne bi bila potrebna: