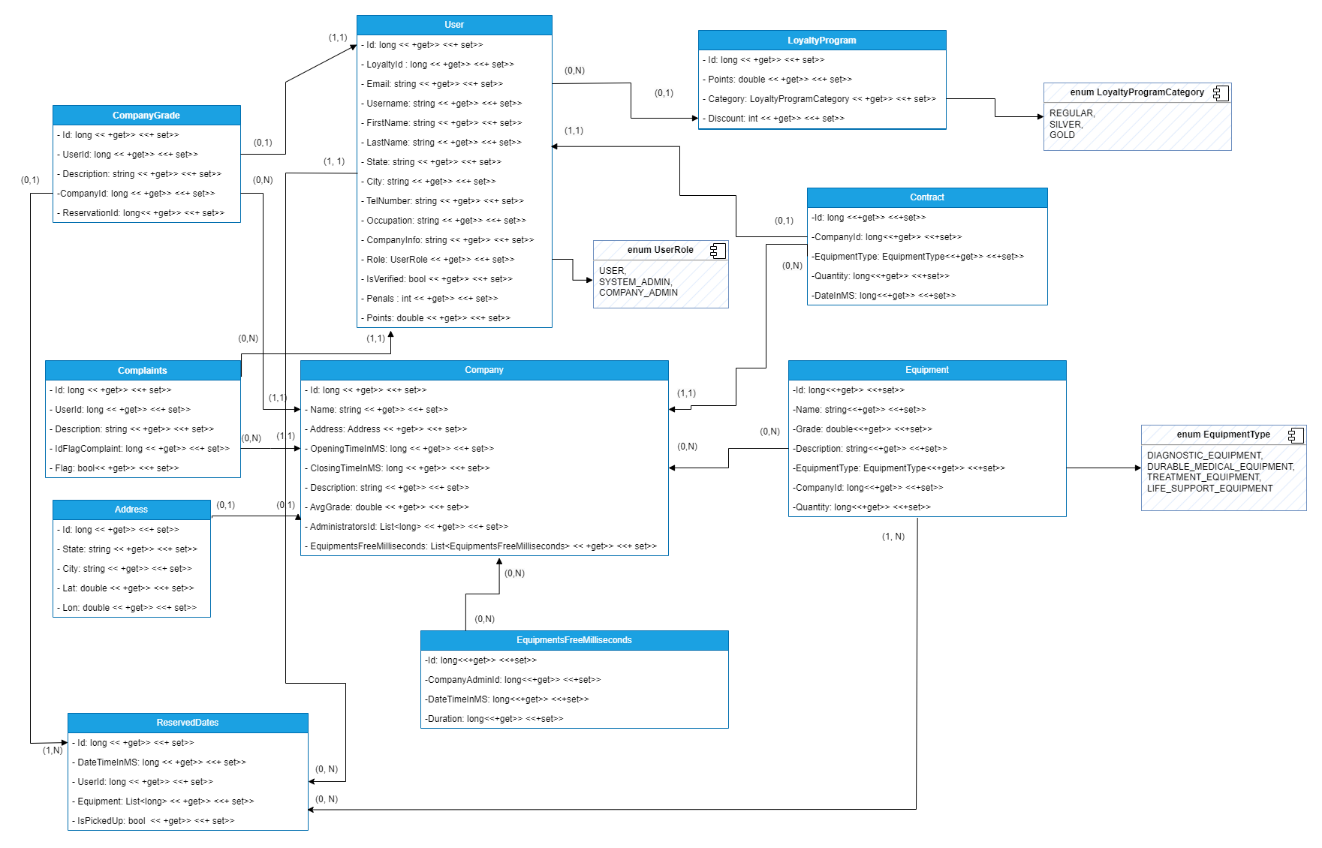
Proof Of Concept

ISA\_2023-tim61

1. Dizajn šeme baze podataka (class diagram):



1. Predlog strategije za particionisanje podataka

Za svaku tabelu u bazi možemo da uradimo vertikalno jer postoje podaci koji se ne menjaju često. Da bismo poboljšali performance možemo da koristimo lazyloading, I za takve podatke bismo radili particionisanje. Na primer, za tabelu Kompanija bismo odvojili podatke kao što su naziv, opis i adresa bile u posebnoj particiji jer se vrlo retko menjaju, dok bi prosečna ocena, predefinisani datumi, itd. Bili na drugoj particiji.   
Horizontalno particionisanje bi nam bilo od velike pomoći jer bismo termine mogli da particionišemo po vremenu. Tako bi nastale particije koje bi sadržale termine za svaki mesec posebno, kao i one koje bi sadržale termine na nedeljnom nivou. Ovobi prikaz radnog kalendara učinilo dosta efikasnijim.

1. Predlog strategije za replikaciju baze i obezbeđivanje otpornosti na greške

Naš najveći problem, trenutno, je taj što naša baza ima *single point of failure*-a , što znači da imamo jednu instance baze, koja, ako otkaže, otkazuje ceo system. To možemo da rešimo tako što ćemo napraviti još neke redudantne instance naše baze podataka I uvezati ih u jedan sistem. Ovo radimo da bismo podigli perfomanse I obezbedili što veću otpornost na greške. Samim tim što smo uradili replikaciju baze, rešili smo problem *single point of failure.* Iskorisitili bismo protokol *ROWA sa primarnom kopijom* koji bi nam omogućio da imamo jednu kopiju koja bi bila “primarna” I ona bi služila sam za čitanje. Za pisanje bi se koristile primarne I sve rezerne kopije koje su raspoložive u tom trenutku. Ako bi primarna kopija postala neoperativna, zamenila bi je neka rezervna koja je unapred određena I samim tim ne bi došlo do gubitka bilo kakvih podataka, jer bi se svaka izmena, koja se desila dok je primarna kopija bila neoperativna , izvršila kada bi se kopija vratila u operativno stanje. Što se tiče alata potrebnih za replikaciju možemo koristiti 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.

1. Predlog strategije za keširanje podataka

Što se tiče keširanja podataka, uveli bismo *“read-throught cache”*  I *“write-throught cache”.* Time bismo u svakom trenutku omogućili da podaci budu konzistenti. Samim keširanjem ubrzavamo rad aplikacije jer pristupamo kešu, a ne samoj bazi. Kao mikroprimer urađen je *“cache-aside”,* ali za celokupnu našu aplikaciju, ovakav tip keširanja ne bi doneo prevelike prednosti. Na primer, često čitamo rezervisane termine. Da bismo ubrazli tako nešto, u kešu koji je tip *“read-throught”* čuvamo podatke o rezervisanim terminima. Time ne bismo opteretili bazu nepotrebnim čitanjem, već bismo samo prvi put učitali u keš, I odatle iščitavali posle.

Sa druge strane bitno je omogućiti i *“CDN”* keširanje koje će da učini da statički podaci budu bliže korisniku, te će se oni brže učitavati kod korisnika, pogotovo zbog velikog broja .js fajlova koje sadrži naša aplikacija. Veliki broj modernih web browser-a podržavaju ovaj vid keširanja, tako da nije potrebno uključivati neki dodatan servis.

1. Okvirna procena za hardverske resurse potrebne za skladištenje svih podataka u narednih 5 godina
   1. Skladištenje korisnika:

Pošto podaci o korisniku zauzimaju u proseku 1.8 KB, a treba čuvati 100 miliona korisnika, biće nam potrebno približno 172 GB.

* 1. Skladištenje kompanija:

Pošto podaci o kompaniji zauzimaju u proseku 3.2 KB, a treba čuvati 100 miliona kompanija, biće nam potrebno približno 306 GB.

* 1. Skladištenje rezervacija:

Pošto podaci o rezervaciji zauzimaju u proseku 2.0 KB, a da je prosečna mesečna kvota oko 500 000 rezervacija, za period od 5 godina biće nam potrebno približno 58 GB.

* 1. Skladištenje opreme:

Pošto podaci o opremi zauzimaju u proseku 1.6 KB, a treba čuvati 100 miliona različite opreme, biće nam potrebno približno 153 GB.

1. Predlog strategije za postavljanje load balansera

S obzirom da se autentifikacija u našoj aplikaciji obavlja preko *JWT*-a (*stateless*), možemo koristiti *“NGINX”* load balanser koji bi koristio *“round robin”* algoritam. Ovakav load balancer može ispuniti naše zahteve za horizontalnim skaliranjem bez uvođenja dodatnih mehanizama za prostiranje sesije između različitih *node*-ova u klasteru. Pošto imamo iste servere, oni će biti podjednako opterećeni. Nemamo potrebe za algoritmima za prepoznavanje sesije u load balanseru, koji bi trošili dodatno vreme, jer koristimo *JWT* za autentifikaciju.

Što se vertikalnog skaliranja tiče, to uvek možemo primeniti bez obzira na arhitekturu sistema tako što ćemo na sve mašine staviti više memorije (64GB zbog veličine naše aplikacije) i jači CPU (trenutno najbolji CPU za high-end workstation sisteme su  AMD Threadripper Pro 5995WX i 5975WX).

1. Predlog koje operacije korisnika treba nadgledati u cilju poboljšanja sistema

Praćenje ponašanja našeg korisnika je esencijalno za dalji napredak aplikacije. Za korisnika možem pratiti koju opremu je često naručivao, od kojih kompanija je najčešće naručivao i tako mu kasnije reklamirati tu opremu ili određenu kompaniju. Što se tiče kompanije, možemo pratiti koja se medicinska oprema najviše iznajmljivala i koji korisnici su najviše koristili usluge te kompanije. Kao dodatne funkcionalosti se mogu uvesti I popusti za određene opreme, bilo to zbog male prodaje ili zbog konkretnog korisnika koji je dosta puta koristio usluge te kompanije. Možemo tim podacima treirati model nenadgledanog mašinskog učenja I tako preporučivati određenu opremu korisniku. Za prikupljanje opštih informacija o našoj aplikaciji (propusnost, zauzeće memorije, iskorišćenje procesora...) možemo koristiti besplatno open-source aplikaciju “*Prometheus”* dok za grafički prikaz prikupljenih podataka možemo iskoristiti aplikaciju “*Grafana”*.

1. Kompletan crtež dizajna predložene arhitekture

