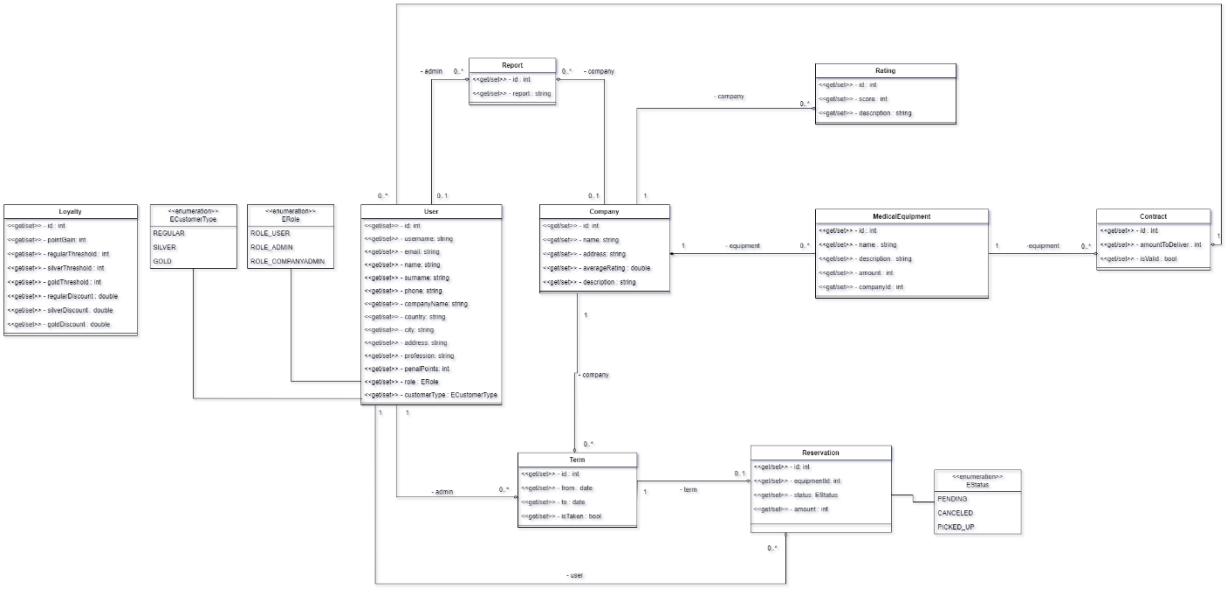
Proof of Concept

Tim 48

1. Dizajn šeme baze podataka



2. Strategija za particionisanje podataka

U našem sistemu mogli bi da primenimo i vertikalno i horizontalno particionisanje podataka.

Što se tiče vertikalnog particionisanja najbolji primer nam daje *Korisnik* gde bih smo mogli da personalne podatke (email, ime, prezime, broj telefona) i addresu (grad, drzavu, adresu, broj) odvojimo u zasebne tabele.

Horizontalno particionisanje bi bilo od velike koristi u slučaju da imamo veliki broj podataka u našoj bazi, te bi smo poboljšali performanse read/write operacija.
Za *korisnika* i *kompanije* mogli bi da iskoristimo list partition i da podelimo po drzavama u kojima se oni nalaze, logika iza ovoga je da kada neko trazi da pokupi opremu, on će tražiti od kompanije koja mu je najbliža. Pošto **PostgreSQL** podržava **subpartitioning** mogli bismo dalje da podelimo po gradovima mada bi ovo bilo overkill za naš sistem.
Što se tiče rezervacija one bi se mogle particionisati po mesecima u godini (Januar, Februar ...) te bih dobavljanje trenutnih rezervacija bilo mnogo brže.
U slučaju još većeg rasta podataka mogli bi smo da shardujemo bazu po geografskim regijama (Evropa, Amerika, Azija, Afrika) mada trenutno nam je ovo overkill.

3. Strategije za replikaciju baze i obezbeđivanje otpornosti na greške

Kako bih napravili pouzdan sistem potrebno je da imamo više instanci baze podataka kako bi otklonili problem koje imamo kada radimo sa samo jednom instancom. Našu mrežu baza bi uvezali u **master-slave** konfiguraciju. **Master** server bi primao samo write zahteve i njega replikuju **slave** serveri koji primaju samo read zahteve, na ovaj način **master** server se rastereti od read zahteva. U slučaju kvara **master** servera za novi **master** server se bira **slave** server čiji su podaci najažurniji.

Što se tiče metode koja bi se koristila za replikovanje **master** servera na **slave** servere, koristili bi robustan sistem koji **PostgreSQL** nudi a to je **WAL (Write-Ahead-Logging) streaming replication**, ova metoda podržava i sinhronu i asinhronu replikaciju pa bi za veću dostupnost najbolje bilo replikacije izvršiti asinhrono. Takođe WAL metoda nam omogućava **Hot Standby** što znači da **slave** server može odradi naše read-only upite dok se vrši replikacija.

4. Strategije za keširanje podataka

Pošto koristimo **Spring**, bilo bi dobro iskoristiti **Hibernate** jer podržava L1 cache po defaultu. Za L2 bi koristili **EhCache** pošto je jedan on najboljih i najpopularnijih provajdera za L2 cache. Najbolje bi bilo da se keširaju podaci o *kompanijama* i o *korisnicima* jer se retko menjaju. Možemo koristiti Write-Behind Caching metodu za ažuriranje L2 cache-a.

Bilo bi i od velike koristi koristiti **CDN** za brzu dostupnost statičkih podataka (JavaScript, HTML, CSS). Centralni server bi bio prisutan za dinamičke podatke dok bi imali edge servere koji bi servirali statičke podatke.

5. Procena za hardverse resurse za narednik 5 godina

Procena potrebne memorije:

- *Korisnik* – Podaci o jednom korisniku zauzimaju oko 765 bajta, za 100 miliona korisnika to iznosi **76.5 GB**
- *Kompanija* – Podaci o jednoj kompaniji zauzimaju oko 557 bajta, nije specificirano ali recimo da imamo 500.000 kompanija to je **0.2785 GB**
- *Medicinska oprema* – Podaci o jednoj stavki za opremu zauzimaju 370 bajta, recimo da za svako od 500.000 kompanija svaka kompanija ima po 100 stavki za opremu to je **18.5 GB**
- *Rezervacija* – Podaci o rezervaciji zauzimaju 41 bajt, ako svakog meseca imamo po 500.000 rezervacija za 5 godina to bi zauzimalo to je **1.23 GB**

Ukupno to daje **96.51 GB**, iako je ovo slobodna procena uvek bi bilo dobro imati više memorije nego što smo procenili.

6. Strategija za postavljanje load balansera

Pošto je naš sistem stateless i koristi **JWT**, nema potrebe da pratimo sesiju i ne treba nam sticky session na našem load balanceru, tako da možemo da koristimo **nginx** i njegovu defaultnu metodu za load balancing a to je **round robin**.

7. Predlog za operacije koje treba nadgledati u cilju poboljšanja sistema

Možemo koristiti **prometheus** sistem za monitoring da bi pratili kako naš sistem radi, to možemo iskoristiti kasnije da vidimo gde bih mogli poboljšati preformanse sistema. Takođe bih mogli pratiti informacije o korisniku, informacije o rezervacijama i na osnovu njih predlagati kompanije koje mu najviše odgovaraju.

Pošto **prometheus** dobro radi sa **grafanom**, koristili bi grafanu za prikaz ovih podataka kako bih dobili bolji vizuelni prikaz.

8. Dizajn arhitekture