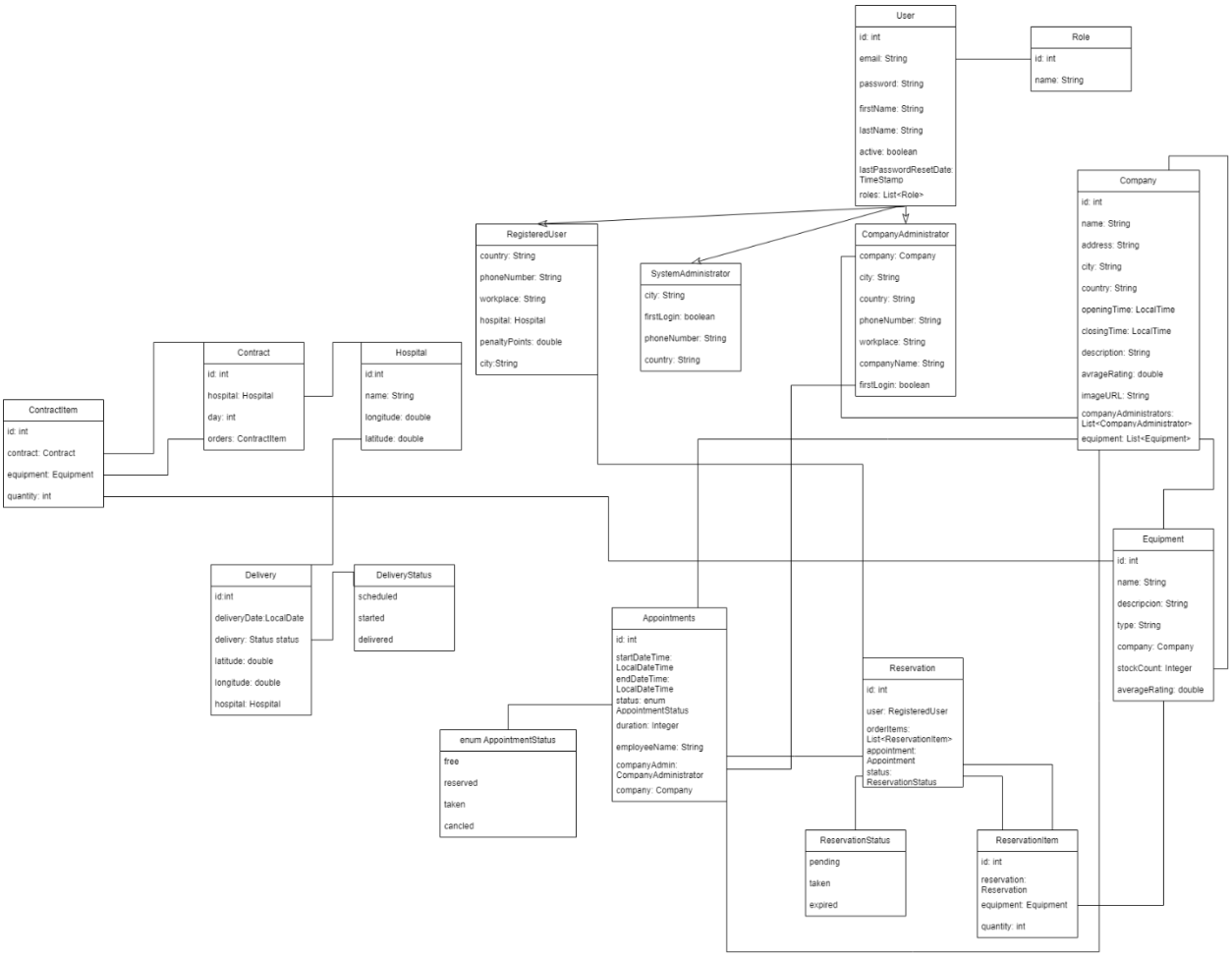
Proof of Concept

1. Dizajn šeme baze podataka



2. Predlog strategije za particionisanje podataka

Horizontalno particionisanje (sharding) omogućava distribuciju podataka na više servera ili čvorova, što olakšava rast sistema. Svaki čvor može rukovati sa svojim deo podataka i zahtevima, umesto da jedan centralni server mora nositi sa svim podacima. Kroz sharding, opterećenje na svakom čvoru se smanjuje, što dovodi do poboljšanja performansi.

Distribucija podataka omogućava i paralelno izvršavanje upita i transakcija, čime se smanjuje vreme odziva sistema. Podaci na primer mogu biti podeljeni po vremenskim periodima ili korisničkim segmentima.

Ako se podaci horizontalno particionišu na različite čvorove ili servere, sistem može postati otporniji na kvarove. Ukoliko jedan čvor doživi problem, ostali čvorovi mogu i dalje pružati usluge bez gubitka svih podataka.

Kada su podaci horizontalno raspoređeni, održavanje i nadogradnja Sistema takođe postaju lakši jer se svaki čvor može ažurirati nezavisno od ostalih. Ovo smanjuje potrebu za isključivanjem celog sistema tokom ažuriranja.

Glavni razlog zašto ova strategija preovladava nad drugim jeste struktura formacije korisnika naše aplikacije. Neki problemi ciljne grupe jesu koncentracija na jednu geografsku povrsinu i nepravilan raspored tipova korisnika.

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

Za replikaciju baze bi se koristila multi-master strategija. Gde bismo imali sinhronu replikaciju za podatke za koje je potrebna velika konzistentnost(npr. rezervacija termina) i asinhronu replikaciju za manje relevantne podatke(npr. korisnikovi podaci o nalogu). Razlog je njena zaštita od failover-a u vidu rezervnih baza sa podacima, kao i mogućnost implementacije za PostgreSQL bazu podataka koja je u trenutnoj aplikaciji. Radi daljeg poboljšanja bi se mogle implementirati dodatne PostgreSQL alatke za rad na klasterima(npr. pgEdge), Health monitoring program za praćenje statusa klastera BP, testiranje kapaciteta BP i nalaženja osetljivih tačaka u sistemu.

4. Predlog strategije za keširanje podataka

TTL (Time-To-Live) Cache je vrsta keširanja koja omogućava postavljanje vremenskog ograničenja na trajanje podataka u kešu i za njegovu realizaciju koristili smo Redis. Razlozi odabira baš TTL Cache strategije jesu:

- Ako se podaci u izvoru često menjaju, a keširane kopije postaju zastarele, TTL Cache omogućava automatsko osvežavanje podataka nakon isteka vremenskog perioda. Na ovaj način se obezbeđuje ažuriranje podataka u kešu i poboljšanje

performansi, memorijom se upravlja efikasnije i smanjen je rizik od zastarenja podataka.

- Ukoliko bi se svaki put kada se traže podaci moralo pristupiti izvoru podataka, moglo bi se povećati opterećenje same baze. Korišćenjem TTL keša, podaci mogu biti privremeno sačuvani lokalno, smanjujući potrebu za stalnim pristupom podacima iz baze.

Redis možemo horizontalno skalirati pomoću topologije implementacije nazvane Redis Cluster što bi značajno dobrinelo radu naše aplikacije sa velikim brojem korisnika.

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

Okvirna procena za predviđene hardverske resurse za potrebe skladištenja podataka u narednih 5 godina je 4.9 terabajta memorije uzimajući i eventualni rast broja korisnika u obzir. Najbolji način za realizaciju datih resursa je horizontalno skaliranje klastera baza podataka.

6. Predlog strategije za postavljanje load balansera

Za strategiju postavljanja load balansera koristila bi se stateless strategija pri distribuciji zahteva klijenta, sa backend klasterom servera. Razlog za implementaciju je efektivnije skladištenje podataka u okviru posebnog klasatera baze, umesto u okviru backend servera, kao I sprečavanja failover-a pri skladištenju na backendu. Takođe je to standard za današnje REST web aplikacije.

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

Operacije korisnika koje bi se trebale nadgledati u cilju poboljšanja same aplikacije, bi definitivno bile one konkurentne prirode i velikog saobraćaja,

naročito zakazivanje termina za rezervacije i preuzimanje opreme, kao i realizacija dostave zbog svoje real-time prirode i količine podataka u protoku.

Kao mikro primer je implementiran monitoring upotrebom Prometheus-a i Grafana-e, gde je napravljen jednostavan alert test nadgledanja CPU performanse procesa našega servera. Naravno dati monitoring se može proširiti velikim brojem use-case scenarija.

8. CRTEŽ DIZAJNA PREDLOŽENE ARHITEKTURE