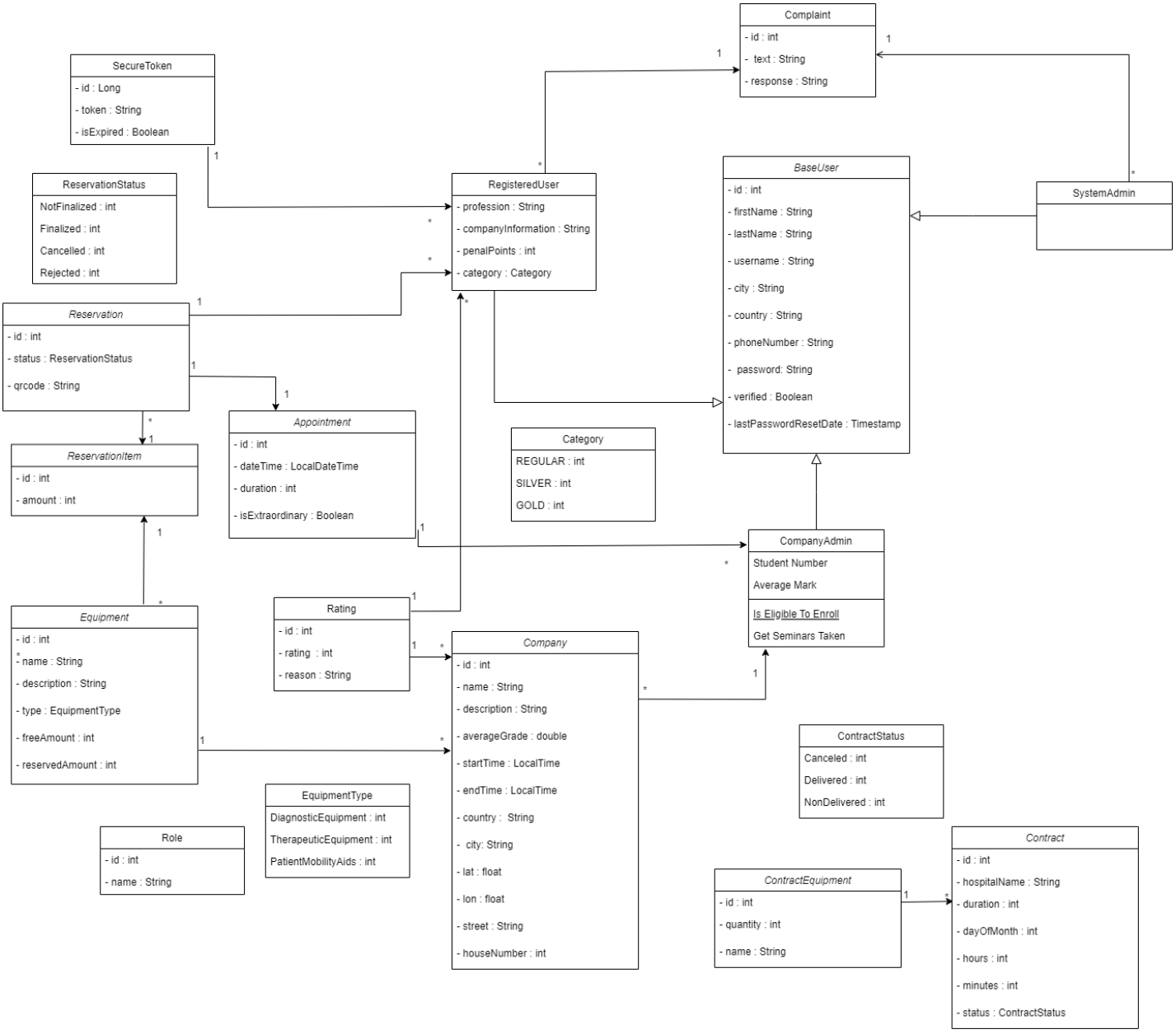
Proof Of Concept

TIM 49 – ISA

1. Klasni dijagram sistema

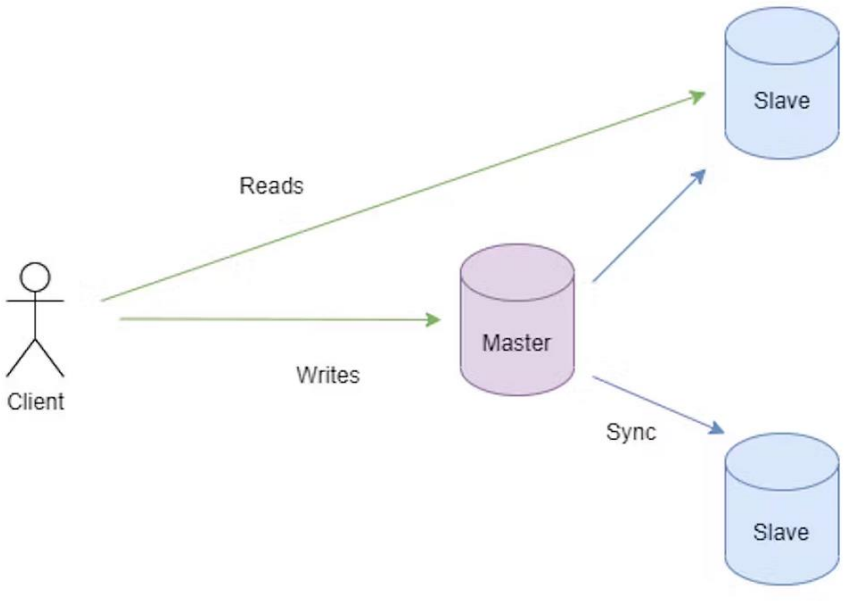


2. Strategija particionisanja podataka

Kada radimo sa tabelama gde nema potrebe za čestim čitanjem nekih podataka uvek je korisno razmotriti primenu particionisanja. Dobar indikator za kreiranje particija je LazyLoading anotacija. Ona nam ukazuje na moguću podelu na 2 tabele, jedne sa primarnim podacima koje izvlačimo iz baze tokom čitanja i druge sa dodatnim podacima. Za kreiranu kompaniju, ili registrovanog korisnika zanimljivo bi bilo kreirati particionisanje na opšte opisne podatke i podatke povezanih entiteta. Dobra ideja bi bila i particionisanje samih termina na osnovu datuma, to jest meseci u godini radi optimizacije kompanijskog kalendara.

3. Strategija za replikaciju baze i obezebeđivanje otpornosti na greške

Potreba za replikacijom baze dolazi iz činjenice da sa samo jednom bazom imamo veliki rizik, a to je da njen pad povlači pad celog našeg sistema. Uvođenjem nekoliko instanci klonova i pokretanjem *primary-secondary (master-slave)* režima rada možemo rešiti ovaj problem.



Svaki read-only zahtev usmeren je ka sluga instancama baze i samim tim master baza dobija manje posla. Obavezna je propagacija podataka ili operacija koje su

dovele do promena radi očuvanja konzistentnosti baze. Master i slave baze u razvoju arhitekture bile bi relacione (obe *PostgreSQL*). Za samu replikaciju kao koristan alat ističu se sam *PostgreSQL Replication* sa *streaming* i logičkom replikacijom , ali i nešto složeniji *Veeam Backup & Replication* koji koristi virtuelnu mašinu sa slikom čitavog sisteam i podržava rad sa *PostgreSQL* bazama.

4. Strategija za keširanje podataka

Za brz pristup informacijama i keširanje podataka koristili smo *In-Memory* bazu podataka *Redis.* Činjenica da ova baza čuva podatke u RAM-u je važna za nas, jer implicira znatno brže operacije čitanja i pisanja. Specificirali smo koje podatke treba smesiti u keš memoriju, ako ih u pretrazi nema unutar keša biće dobavljeni klasičnim čitanjem iz baze i smešteni u keš. Redis je baziran na sistemu ključa i vrednosti, gde svaki podatak ima jedinstven ključ. Može se postaviti određeno vremensko ograničenje (ttl-time-to-live) na ključeve u Redis kešu i sitem će automatski ukloniti ključ iz keša nakon isteka vremenskog perioda.

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

Ako pogledamo kako smo modelovali korisnika u sistemu možemo da vidimo sledeće klase:

Base user:

Naziv promenljive	Tip promenljive	Zauzeće memorije	Prosečno zauzeće
Id	unsigned int	4 bytes	4 bytes
City	Char(255)	1 byte * 255	12 bytes
Country	Char(255)	1 byte * 255	12 bytes
First name	Char(255)	1 byte * 255	12 bytes
Last name	Char(255)	1 byte * 255	20 bytes
Last password reset date	DateTime	32 bytes	24 bytes
Password	Char(255)	1 byte * 255	70 bytes
Phone	Char(255)	1 byte * 255	12 bytes
Username	Char(255)	1 byte * 255	12 bytes
Verified	boolean	1 bit	1 bit
			Ukupno: 178 bytes

Registered user:

Naziv promenljive	Tip promenljive	Zauzeće memorije	Prosečno zauzeće
Category	Integer	4 bytes	4 bytes
Company info	Char(255)	1 byte * 255	0*
Penal points	Integer	4 bytes	4 bytes
Profession	Char(255)		0*
Id	Unsigned int	4 bytes	4 bytes
			Ukupno: 16 bytes

* Ova polja nismo koristili u projektu pa ih nismo ni računali u zauzeću memorije

System admin:

Naziv promenljive	Tip promenljive	Zauzeće memorije	Prosečno zauzeće
Id	Unsigned int	4 bytes	4 bytes
			Ukupno: 4 bytes

Company admin:

Naziv promenljive	Tip promenljive	Zauzeće memorije	Prosečno zauzeće
Id	Unsigned int	4 bytes	4 bytes
Company Id	Unsigned int	4 bytes	4 bytes
			Ukupno: 8 bytes

Kada uzmemo u obzir raspodelu korisnika, recimo da imamo sledeću situaciju:

- Administrator – 2%
- Company admin – 18%
- Registered user – 80%

Na osnovu ovih podataka možemo da izračunamo prosečno zauzeće memorije:

100 miliona * (178 bytes + (16 bytes * 80/100) + (4 bytes * 2/100) + (8 bytes * 18/100)) = 17.91 GB

Sada da vidimo koliko bi zauzimale kompanije,

Company:

Naziv promenljive	Tip promenljive	Zauzeće memorije	Prosečno zauzeće
Id	unsigned int	4 bytes	4 bytes
Street	Char(255)	1 byte * 255	30 bytes
House number	Integer	4 bytes	4 bytes
City	Char(255)	1 byte * 255	12 bytes
Country	Char(255)	1 byte * 255	12 bytes
Name	Char(255)	1 byte * 255	20 bytes
Description	Char(255)	1 byte * 255	120 bytes
Start time	LocalTime	16 bytes	16 bytes
End time	LocalTime	16 bytes	16 bytes
Latitude	Real	8 bytes	8 bytes
Longitude	real	8 bytes	8 bytes
			Ukupno: 260 bytes

Ako napravimo aproksimaciju da kompanije u proseku imaju 3 administratora dolazimo do broja:

18/100/3 * 100miliona * 260 bytes = 1.45GB

Pređimo sada na zauzeće memorije rezervacija.

Appointment:

Naziv promenljive	Tip promenljive	Zauzeće memorije	Prosečno zauzeće
Id	unsigned int	4 bytes	4 bytes
Date time	DateTime	32 bytes	32 bytes
Duration	Integer	4 bytes	4 bytes
Company admin id	Unsigned int	4 bytes	4 bytes
			Ukupno: 44 bytes

Reservation:

Naziv promenljive	Tip promenljive	Zauzeće memorije	Prosečno zauzeće
Id	unsigned int	4 bytes	4 bytes
Amount	Integer	4 bytes	4 bytes
Equipment id	Unsigned int	4 bytes	4 bytes
Reservation id	Unsigned int	4 bytes	4 bytes
			Ukupno: 16 bytes

Reservation item:

Naziv promenljive	Tip promenljive	Zauzeće memorije	Prosečno zauzeće
Id	unsigned int	4 bytes	4 bytes
Date time	DateTime	32 bytes	32 bytes
Duration	Integer	4 bytes	4 bytes
Company admin id	Unsigned int	4 bytes	4 bytes
			Ukupno: 44 bytes

Equipment:

Naziv promenljive	Tip promenljive	Zauzeće memorije	Prosečno zauzeće
Id	unsigned int	4 bytes	4 bytes
Description	Char(255)	1 byte * 255	32 bytes
Free amount	Integer	4 bytes	4 bytes
Name	Unsigned int	1 byte * 255	20 bytes
Reserved amount	Integer	4 bytes	4 bytes
Type	Integer	4 bytes	4 bytes
Company id	Unsigned int	4 bytes	4 bytes
			Ukupno: 72 bytes

Svaki mesec imamo 500 hiljada porudžbina, što znači da sigurno imamo 500 hiljada *Appointment-a*, a recimo da 1% svih *Reservation-a* bude otkazan i da svaka porudžbina ima prosečno 5 artikala. Sa vim pretpostavkama dobijamo sledeću računicu:

500 hiljada * 12 * 5 * (44 bytes + 101/100 * 16 bytes + 5 * 44 bytes) = 7.82GB

Ako još svaka kompanija u proseku ima 3 hiljade instanci opreme u ponudi dolazimo do toga da će oprema zauzimati dodatnih:

18/100/3 * 100 miliona * 3 hiljade * 72 bytes = 1207GB

Kada sve zbrojimo dolazimo do brojke od:

17.91 B + 1.45GB + 7.82GB + 1207GB = 1234.19GB

6. Predlog strategija za postavljanje *load balancera*

S obzirom da se za autentifikaciju korisnika za pristup našoj aplikaciji koristi *JWT* (*JSON Web Token*) koji je *stateless* ne moramo da brinemo o međusobnoj komunikaciji servera i praćenju sesija korisnika. Takođe ako neki server zbog kvara prestane sa radom, osim dodatnog opterećenja drugih servera, koje ne možemo da izbegnemo, ne gubimo nikakve podatke.

Za *load balancer* možemo koristiti *HAProxy* load balancer koji je besplatan i open source. *HAProxy* koriste *GitHub, Reddit, Airbnb*... *HAProxy* nudi više različitih *load balancing* algoritama(*roundrobin, leastconn, source*) od kojih bi nama odgovarao default i najjednostavniji *roundrobin*.

Takođe ako naša aplikacija bude to zahtevala moguće je koristiti i geografsko raspoređivanje opterećenja, čime bismo postigli manji *latency* za našu aplikaciju.

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

Praćenje operacija korisnika je ključno za sve korisnike u sistemu. Administratori dobijaju informacije o tome koje radnje korisnika traju najduže i mogu da optimizuju te radnje, administratori kompanije bi mogli da prave reklame na osnovu praćenja ponašanja korisnika na sajtu, dok bi korisnici mogli da uz manje muke, korišćenjem reklama, pronaću opremu koja im treba. Od svih operacija na sajtu za nadgledanje bih izdvojili sledeće:

- Prikaz i pretraživanje kompanije
- Prikaz i pretraživanje opreme
- Zakazivanje termina preuzimanja opreme

Za prećenje aktivnosti korisnika na našem sajtu možemo koristiti *Google analytics*.

8. Kompletan crtež dizajna predložene arhitekture