Proof of concept

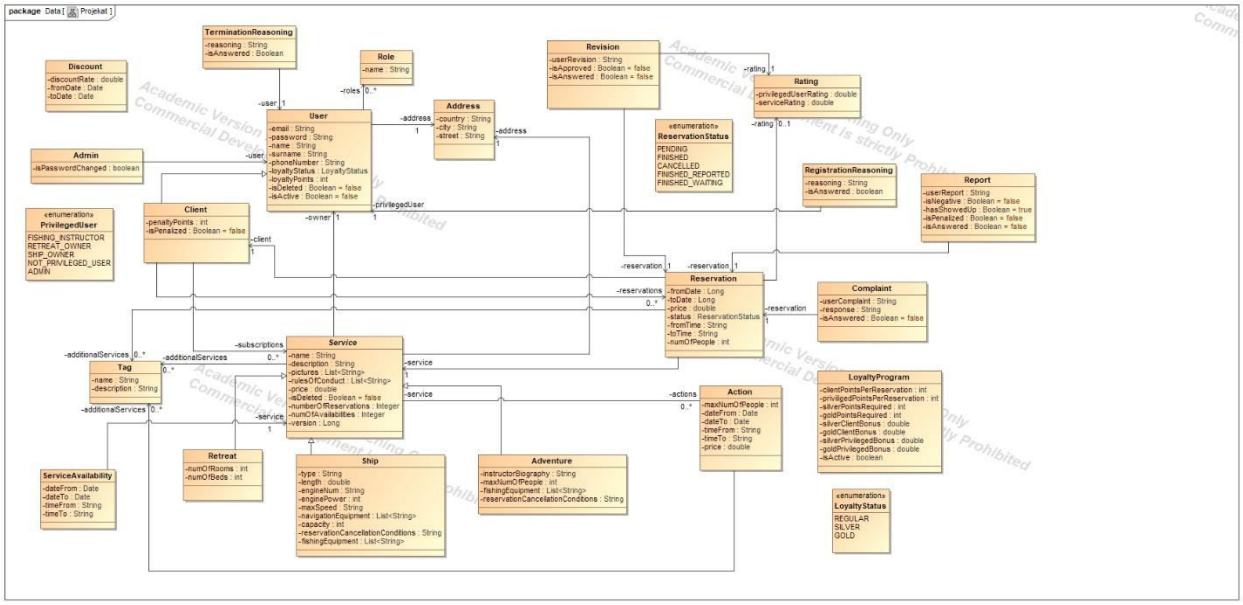
Projekat ISA-MRS, tim 3

Teodor Sakal Francišković, SW 22/2019

Matija Zarić, SW 24/2019

Zoran Bukorac, SW 40/2019

1. Dizajn šeme baze podataka



2. Predlog strategije za particionisanje podataka

Zbog predviđenog broja od 100 miliona korisnika, došlo bi do potrebe smeštanja velike količine podataka u bazu što bi znatno usporilo čitav sistem. Particionisanje se može vršiti horizontalno i vertikalno. Generalno postoje podaci koji se čitaju zajedno sa podacima koji se traže, a nisu uvek potrebni, tako da bi se moglo izvršiti particionisanje. Svaku tabelu, kod koje postoje podaci koji su potrebni prilikom svakog čitanja, možemo podeliti na više tabela, kod kojih će postojati jedna sa podacima, koji se uvek čitaju, a u ostalim tabelama bi bili podaci koji se čitaju u zavisnosti od zahteva.

Tabelu rezervacija možemo horizontalno particionisati, tako što bi se mogla podeliti na više tabela u zavisnosti od meseca ili sedmice, kako bi se i lakše pretraživala dostupnost servisa, a takođe i brže pravili izveštaji koji su potrebni vlasniku.

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

Trenutno najveći problem naše baze je to, što ceo sistem zavisi od nje, tj. ako se desi da otkaže baza otkazuje i ceo sistem. Ovaj problem se može rešiti uvođenjem više redundantnih baza i uvezati ih u standardnu *master-slave* konfiguraciju. Uvođenjem ovih baza, svi zahtevi koji vrše samo čitanje će se slati na pomoćne (*slave*) baze, dok će se svi zahtevi, koji zahtevaju nekakav upis u bazu, slati glavnoj bazi podataka. Način za očuvanje konzistentnosti baze može biti propagacija akcija. Glavna i pomoćne baze mogu biti instance naše već postojeće *Postgres* baze, s obzirom da je ona relaciona i samim tim struktuirana i laka za održavanje.

4. Predlog strategije za keširanje podataka

Kako je L1 keširanje interno podržano unutar sesije od strane *Hibernate-a,* on je uključen u našu aplikaciju. Što se tiče L2 keširanja, *Hibernate* podržava i njega, ali je potreban neki eksterni provajder, a mi smo se odlučili za *EhCache*. *EhCache* podržava sve strategije keširanja, a ujedno je jedan od najboljih i najkorišćenijih provajdera za L2 keširanje u *Spring* aplikacijama. Ideja je keširati podatke koji se relativno često koriste, a retko menjaju. U našem slučaju to bi mogli biti podaci o vikendicama, brodovima i avanturama sistema, ali bi se možda morao modifikovati način obezbeđivanja sigurnog rezervisanja entiteta prilikom istovremenog pokušaja rezervacije istih od strane dva klijenta.

Takođe, bitno je omogućiti i CDN keširanje koje omogućuje keširanje statičkih podataka bližih korisniku. Što se tiče ove vrste keširanja, veliki broj modernih pretraživača ga podržava pa nije potrebno uključivati neki dodatni servis.

Što se tiče tehnika izbacivanja sadržaja iz keša, unutar mikro primera keširanja u našoj aplikaciji, odlučili smo se za zadavanje TTL vrednosti, odnosno vremena koje objekat može da provede u kešu bez obzira na pristupanje istom. Možda ovo nije najbolje rešenje za veliki broj korisnika i veliku posećenost sajtu, pa bi bilo bolje iz keša izbacivati najranije korišćene podatke ili zadati vrednost TTI, odnosno vreme koje objekat može provesti u kešu, a da mu u tom vremenu niko ne pristupa.

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

Korisnici – klijenti prosečno zauzimaju 2,4 KB u bazi, dok ostali korisnici prosečno zauzimaju 1,2 KB, ako bismo računali da je procenat vlasnika i administratora u sistemu 30% na 100 miliona korisnika, korisnici zauzimaju oko 204 GB, zaokruženo na oko 200 GB.

Entiteti sistema – podaci o entitetima prosečno zauzimaju oko 30 KB, ako pretpostavimo da svaki vlasnik ima najviše 2 entiteta i da je procenat vlasnika u sistemu 20%, onda dobijamo da entiteti zauzimaju oko 1 TB.

Rezervacije – svaka rezervacija u bazi zauzima oko 3 KB, ako pretpostavimo da na mesečnom nivou imamo milion rezervacija, dobijamo da one na mesečnom nivou zauzimaju oko 3 GB, što je na petogodišnjem nivou oko 180 GB.

Žalbe – svaka žalba zauzima oko 3 KB, a recimo da za 20% rezervacija postoji žalba, onda zauzeće na petogodišnjem nivou iznosi oko 40 GB.

Izveštaji – svaki izveštaj o rezervaciji zauzima oko 2 KB, ako svaka rezervacija ima izveštaj onda na petogodišnjem nivou dobijamo zauzeće koje iznosi oko 120 GB.

Ako saberemo dobijeno zauzeće za važnije entitete sistema, na petogodišnjem nivou dobijamo zauzeće koje iznosi oko 1,5 TB, ali treba takođe uračunati i ostale entitete sistema,

pa na osnovu toga pretpostavljamo da će nam na petogodišnjem nivou za skladištenje resursa biti potrebno oko 2,5 TB memorije.

6. Predlog strategije za postavljanje *load balancera*

S obzirom na veliki broj zahteva, vremenom jedan server ne bi bio dovoljan da isprati sve zahteve i pruži dobre performanse našoj aplikaciji. Ovo se može rešiti horizontalnim skaliranjem, tj. povećao bi se broj servera koje naša aplikacija koristi. Zbog ovoga potrebno je postojanje *Load balancera*, koji će da odlučuje kojem će serveru poslati koji zahtev. Tehnika raspodele, za koju smo se odlučili, jeste *Round Robin*, jer je relativno jednostavan za implementaciju, a pruža dobre performanse i ravnomerno raspoređuje zahteve na sve servere. Takođe u našoj aplikaciji za autentifikaciju je korišćen JWT, čime se na server ne pamte podaci o ulogovanim korisnicima, pa se tako zahtevi istog korisnika mogu raspoređivati na različite servere.

7. Predlog koje operacije korisnika treba nadgledati radi poboljšanja sistema

Nadgledanje korisničkog rada je jako bitno za razvoj i poboljšanje aplikacije. Samim tim jako je važno nadgledanje svih operacija korisnika koje imaju veze sa rezervisanjem entiteta, pretragom entiteta za rezervaciju, rezervisanje entiteta na akciji, kao i operacije koje bi vlasnik izvršavao vezane za njegov entitet, poput dodavanja akcija, definisanja termina dostupnosti za servis, kao i samu izmenu podataka svog entiteta.

8. Dizajn predložene arhitekture