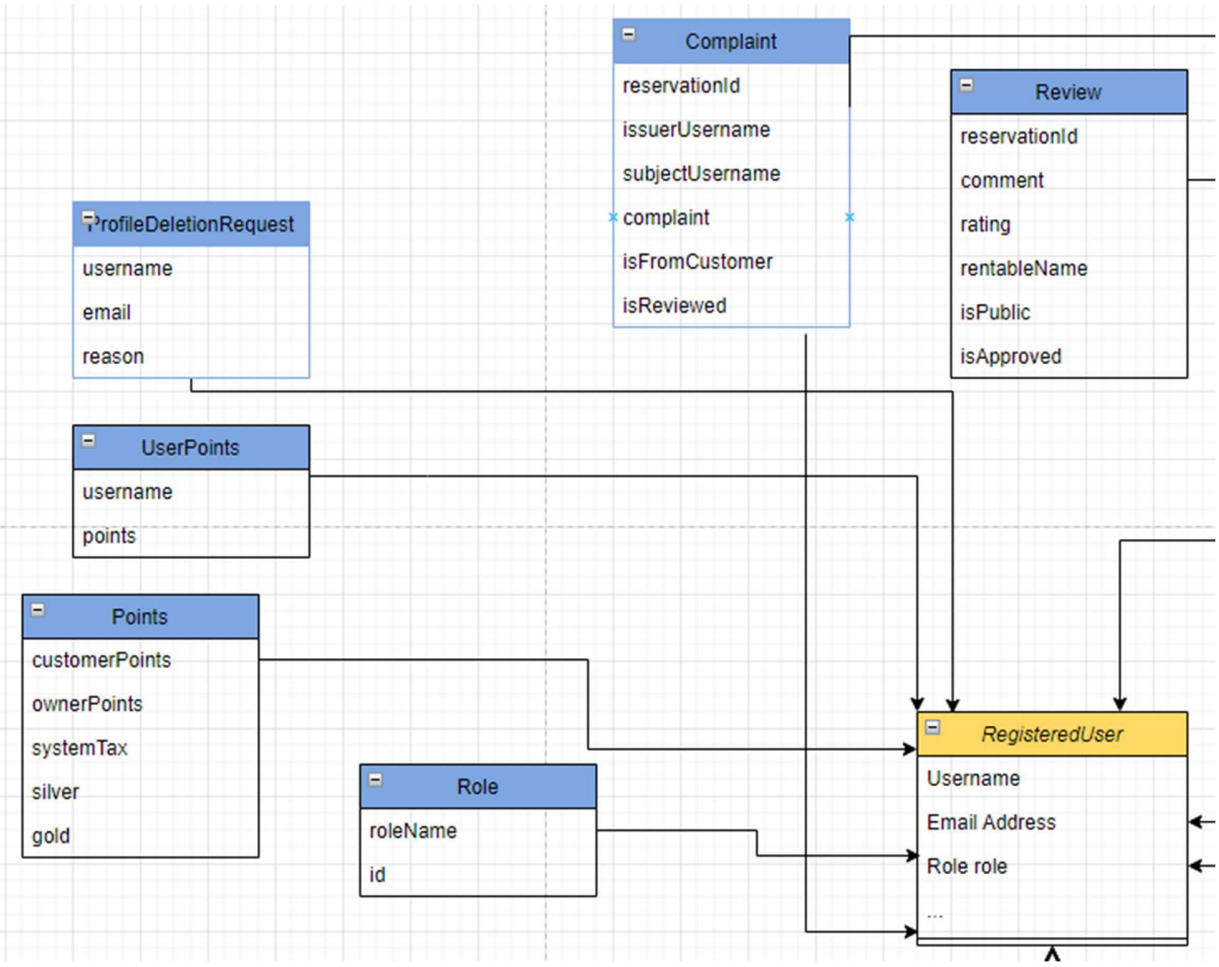
Mihajlo Kisić Luka Miletić Filip Pinjuh

Proof of concept

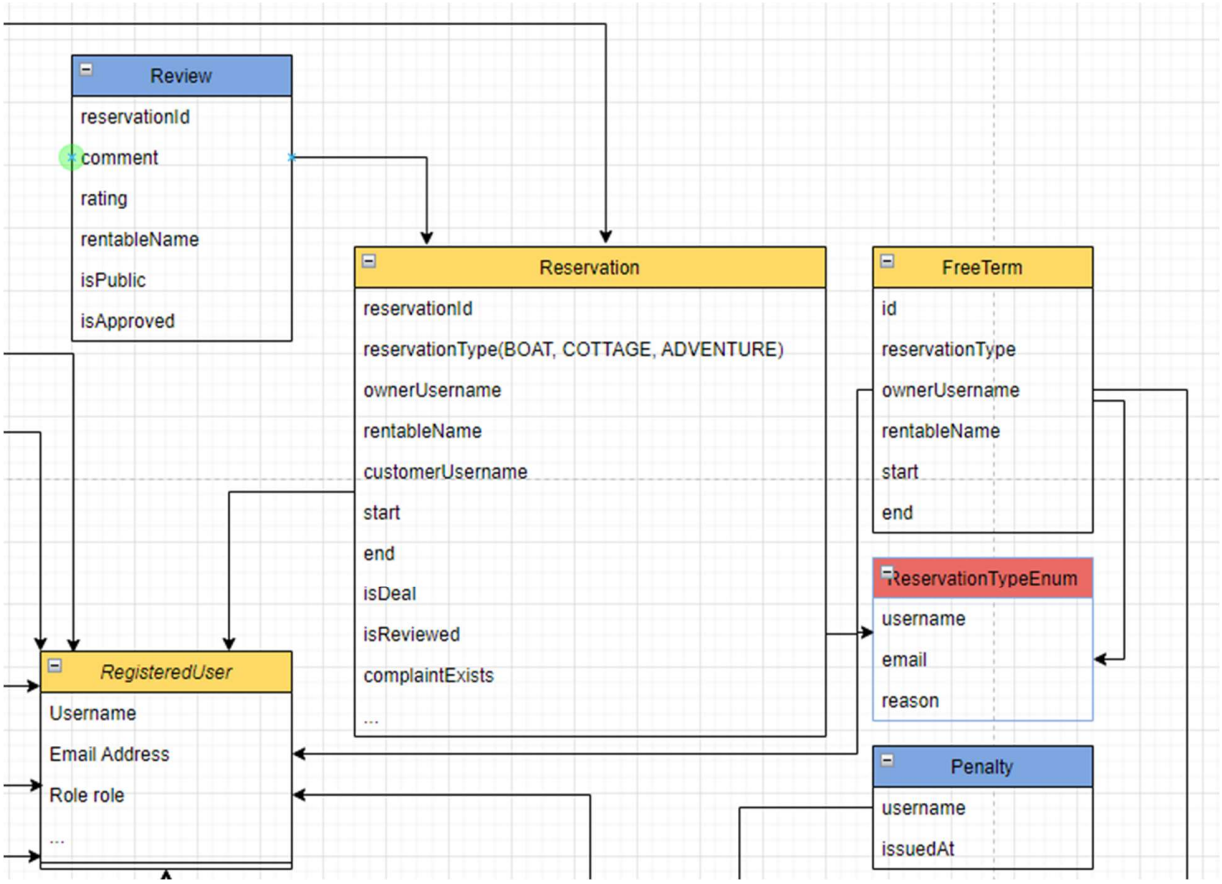
Dizajn šeme baze podataka

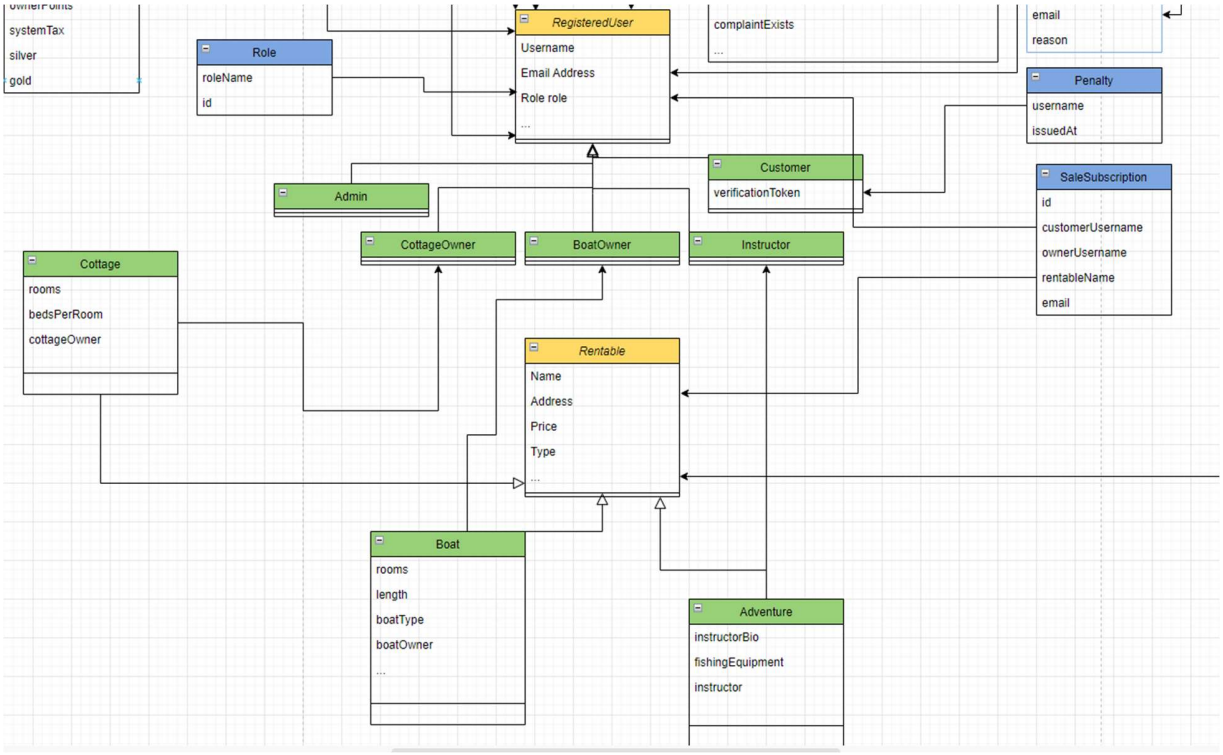
Dizajn šeme baze podataka predstavljen je pomoću klasnog dijagrama pomoću kog je dizajniran model projekta.

Na sledećoj slici prikazane su ostale klase potrebne za implementaciju ostatka funkcionalnosti:



Na sledećoj slici prikazano je nasleđivanje roditeljskih klasa koje su olakšale rukovanje podacima u bazi podataka:


Review
reservationId
comment
rating
rentableName
isPublic
isApproved
Reservation
reservationId
reservationType(BOAT, COTTAGE, ADVENTURE)
ownerUsername
rentableName
customerUsername
start
end
isDeal
isReviewed
complaintExists
...
FreeTerm
id
reservationType
ownerUsername
rentableName
start
end
ReservationTypeEnum
username
email
reason
RegisteredUser
Username
Email Address
Role role
...
Penalty
username
issuedAt


systemTax
silver
gold
Role
roleName
id
RegisteredUser
Username
Email Address
Role role
...
complaintExists
...
email
reason
Penalty
username
issuedAt
Customer
verificationToken
SaleSubscription
id
customerUsername
ownerUsername
rentableName
email
Admin
CottageOwner
BoatOwner
Instructor
Cottage
rooms
bedsPerRoom
cottageOwner
Rentable
Name
Address
Price
Type
...
Boat
rooms
length
boatType
boatOwner
...
Adventure
instructorBio
fishingEquipment
instructor

Mihajlo Kisić Luka Miletić Filip Pinjuh

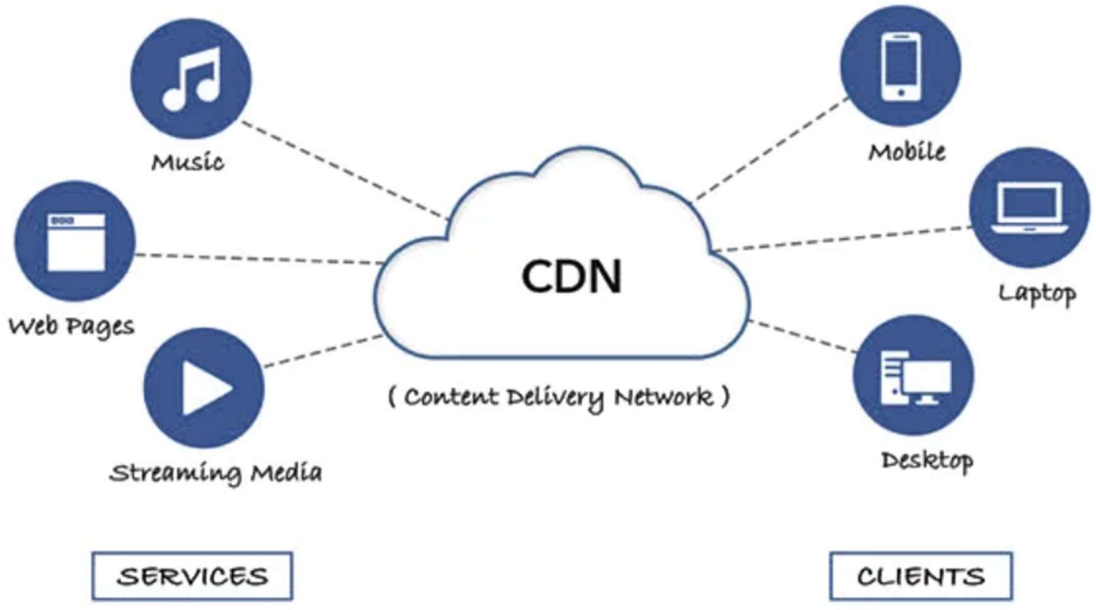
Predlog strategije za keširanje podataka

Povećanjem broja korsnika koji koriste nas sistem količina podataka i poziva ka bazi nesrazmerno se povećava. Ovo usporava rad aplikacije, time čini korisničko iskustvo nepoželjnim.

Ovaj proces najbolje je implementirati za keširanje primarno statičkih podataka, kao što su podaci o korisnicima, vikendicama, brodovima, avanturama, a najznačajnije rad sa slikama. Tokom korišćenja ovi podaci se mnogo manje shodni promenama, za razliku od slobodnih termina, rezervacija (akcija), slobodnih dana … Ipak i ove podatke možemo čuvati u keš memoriji, uz to što kada pravimo promene nad ovim podacima u bazi morali bi da ih menjamo i u keš memoriji. Ukoliko ovu strategiju uparimo sa Event Sourcing-om mogli bi da optimizujemo koje podatke čuvamo u keš-u pogotovo zbog ograničene keš memorije.

Ukoliko se keš napuni, dobra strategija je LRU – Last Recently Used, kojom se izbacuju najstariji dobavljeni podaci.

CDN (Content Delivery Network) keširanje bi takođe bilo povoljno implementirati, kako će ono pomoći prilikom bržeg dobavljanja statičkih podataka, uključujući slike, i veb stranice. Kako je posrednik između podataka i klijenta u ovom slučaju proxy server lociran korisniku bliže u odnosu na originalan server aplikacije, očekivano je da će i traženi podaci stići brže. Ova implementacija posebno je korisna ukoliko uvedemo pretpostavku da se shodno velikom broju korisnika (100 miliona) naša aplikacija koristi internacionalno. Aplikacija FishingBooker trenutno implementira na mikro primeru keširanje često dobavljenih DTO podataka za prikaz lista svih entiteta. Korišćen je EnCache, kao čest izbor za keširanje u Spring Boot aplikacijama.



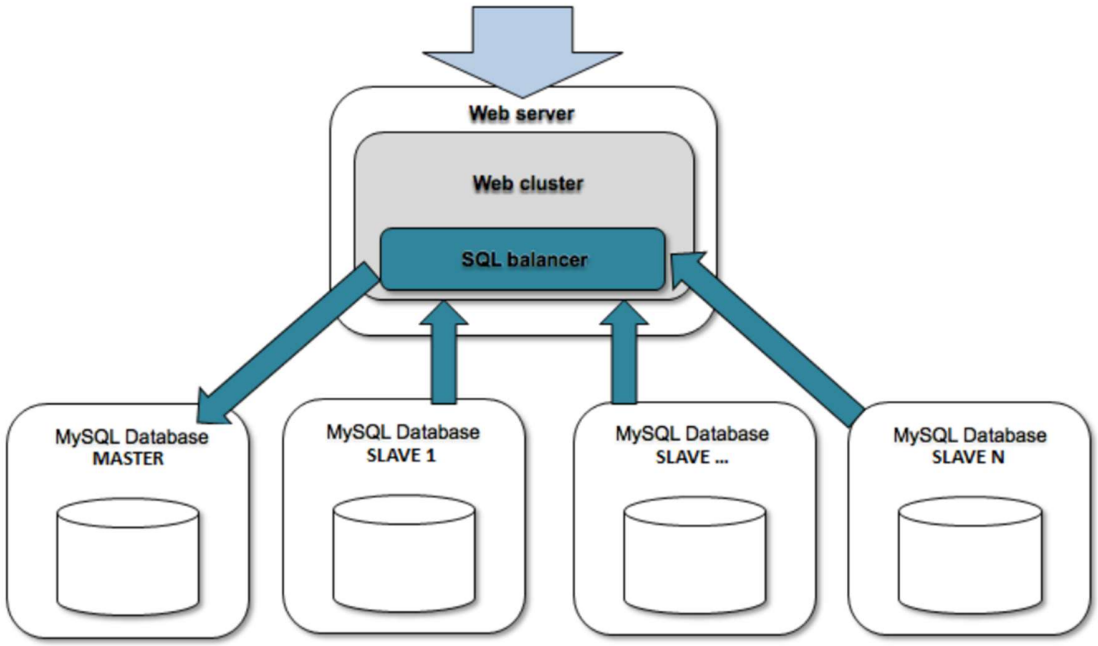
Mihajlo Kisić Luka Miletić Filip Pinjuh

Predlog strategije za replikaciju baze i obezbeđivanje otpornosti na greške

Tehnika koju možemo da upotrebimo za povećanje brzine aplikacije je da zamenimo server koji koristimo sa mnoštvom servera koji bi radili po maste-slave principu. Korisnici, kao i ostali entiteti koje upotrebljujemo bi bili rasporedjeni po serveri na osnovu geografske lokacije, bilo to po državama ili još ambicioznije kontinentima gde bi svaki kontinent imao broj servera srazmeran sa korisnicima na tom kontinentu.

Takođe bismo uveli i replikaciju podataka. Koristili bi master i slave baze. Podaci bi se nalazili na master bazi i na nju bi se vršilo pisanje, dok bi se kopije nalazile na slave bazama. U sistemu bi se radila asinhrona sinhronizacija podataka između master i slave baza kako se ne bi mreža opterećivala. Takođe ukoliko bi došli u situaciju da se master baza previše preopterećuje, zahtevi za čitanje podataka iz baze bi se slali ka slave bazama. Ukoliko bi došlo do otkaza master baze rad aplikacije se nastavlja korišćenjem slave baze.

Na ovaj način bi mogli na velikoj skali osigurati otpornost našeg sistema na greške.



Mihajlo Kisić Luka Miletić Filip Pinjuh

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

Pretpostavke sistema:

- Ukupan broj korisnika: 100 miliona
- Broj rezervacija na mesečnom nivou: milion
- Skalabilan i visoko dostupan sistem

Zauzetost memorije:

- U proseku svaki entitet zauzima oko 2Kb
- Slike zauzimaju oko 500Kb

U proseku jedan korisnik uz sve entitete koje kreira zauzima 2Mb

Za 100 miliona korisnika potrebna je memorija od 300GB

Milion rezervacija mesečno potrebno je 50Gb

Za ocene, žalbe, zahteve 20Gb

Računajući da je za godinu dana rada sistema potrebno oko 1Tb prostora:

Pretpostavljamo da je za 5 godina potrebno oko 5 Tb prostora.

Mihajlo Kisić Luka Miletić Filip Pinjuh

Predlog strategije za particionisanje podata

U mnogim rešenjima velikih razmera, podaci su podeljeni na particije kojima se može zasebno upravljati i kojima se može pristupiti. Particioniranje može poboljšati skalabilnost, smanjiti svađe i optimizovati performanse. Takođe može pružiti mehanizam za podelu podataka prema obrascu upotrebe.

Međutim, strategija podele mora biti pažljivo odabrana kako bi se maksimizirale prednosti uz minimaliziranje štetnih učinaka.

Postoje dve tipične strategije za particioniranje podataka:

Horizontalno particionisanje. U ovoj strategiji svaka je particija zasebno skladište podataka, ali sve particije imaju istu šemu. Svaka particija poznata je kao deonica i sadrži određeni podskup podataka, kao što su sve narudžbine za određeni skup kupaca. Ovakvu strategiju bi koristili u slučajevima gde tabele imaju mnogo redova i želimo da učinimo da se može efikasno skalirati sa porastom podataka. Ovim bi se takođe smanjilo vreme potrebno za indeksiranje i ubrzale operacije nad podacima. U našem slučaju, imalo bi smisla deliti tabelu sa rezervacijama na ovaj način, imajući u vidu da je to ključni aspekt sistema i direktno utiče na opterećenost sistema.

Vertikalno particionisanje. U ovoj strategiji, svaka particija sadrži podskup polja za stavke u pohrani podataka. Polja su podeljena prema obrascu korišćenja. Na primer, polja kojima se često pristupa mogu se smestiti u jednu vertikalnu particiju, a polja kojima se ređe pristupa u drugu. U našem slučaju, rezervacije imaju informacije poput početka, kraja, cene, dodatnih usluga, broja gostiju. Podaci kojima se najčešće pristupa su početak i kraj rezervacije zbog provere zauzetosti i onemogućavanja preklapajućih rezervacija. Imajući ovo u vidu, optimalno bi bilo vertikalno particionisati u jednu particiju informacije o početku i kraju rezervacije, a u drugu informacije o ceni, gostima i dodatnim uslugama.

Predlog koje operacije korisnika treba nadgledati u cilju poboljšanja rada sistema

Jedan od najvažnijih osobina svake aplikacije je visoka operabilnost, usability i pounda baš onih usluga koje su neophodne klijentima. Da bi se ovo uspešno postiglo, neophodno je na neki način pratiti korišćenje aplikacije od strane korisnika i uočavanje eventualnih mana u implementaciji i nedostataka u funkcionalnostima koje bi koristile klijentima. Efikasan način da se ovako nešto realizuje je Event sourcing.

On podrazumeva efikasno i usmereno beleženje akcija korisnika na raznim delovima sistema. U ovo može da spada dužina određenih akcija, njihova sekvenca, praćenje njihove uspešnosti ili broja otkaza. U našem slučaju, najvažnije je da se prati tok rezervisanja entiteta kroz više koraka. Koristili bi se logovi koji bi prikazivali vreme ulaksa u svaki korak rezervacije, zadržavanje na njemu, vraćanje na prethodne korake kao i odustanak od rezervacije. Ovim bi se vremenom mogla oformiti jasnija slika o nedostatku u procesu rezervcije i dovesti do neophodnih izmena.

Mihajlo Kisić Luka Miletić Filip Pinjuh

Predlog strategije za postavljanje load balansera

Postavljamo load balancer izmedju servera i klijenata, on će nam omogućiiti da adekvatno raspodelimo zadatke izmedju servera. Jedan od algoritama koji možemo da koristimo je “Least Pending Request”.

Ovaj algoritam prati broj zahteva poslatih ka svakom serveru. Broj zahteva je srazmeran sa potrebnim vremenom da se isti izvrše. Dakle novopristigli zahtev šaljemo serveru koji ima najmanji red zahteva - red čekanja. Algoritam može efikasno da se nosi sa kašnjenjem servera.

Ovaj algoritam handluje prepoznavanje zahteva i odredjivanje njihovog rasporedjivanja po serverima.

KOMPLETAN CRTEŽ DIZAJNA PREDLOŽENE ARHITEKTURE