PROOF OF CONCEPT

SADRŽAJ

Fishing Booker

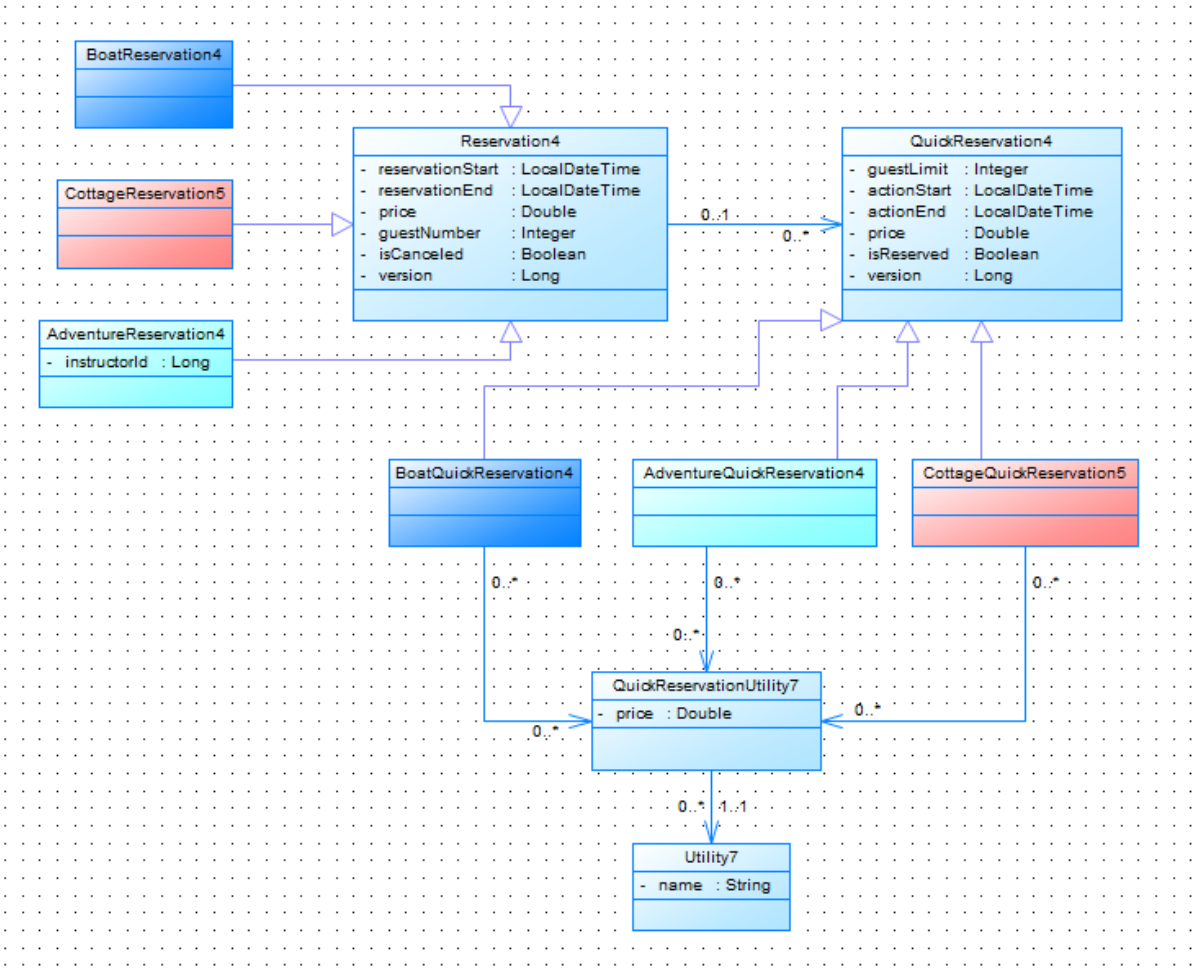
DIZAJN ŠEME BAZE PODATAKA

Dizajn šeme baze podataka predstavljen je pomoću klasnog dijagrama pomoću koga je ista i generisana (podsredstvom *Spring Boot*-a). U pratećem folderu nalazi se PowerDesigner projekat, u kome je napravljeno 7 posebnih dijagrama.

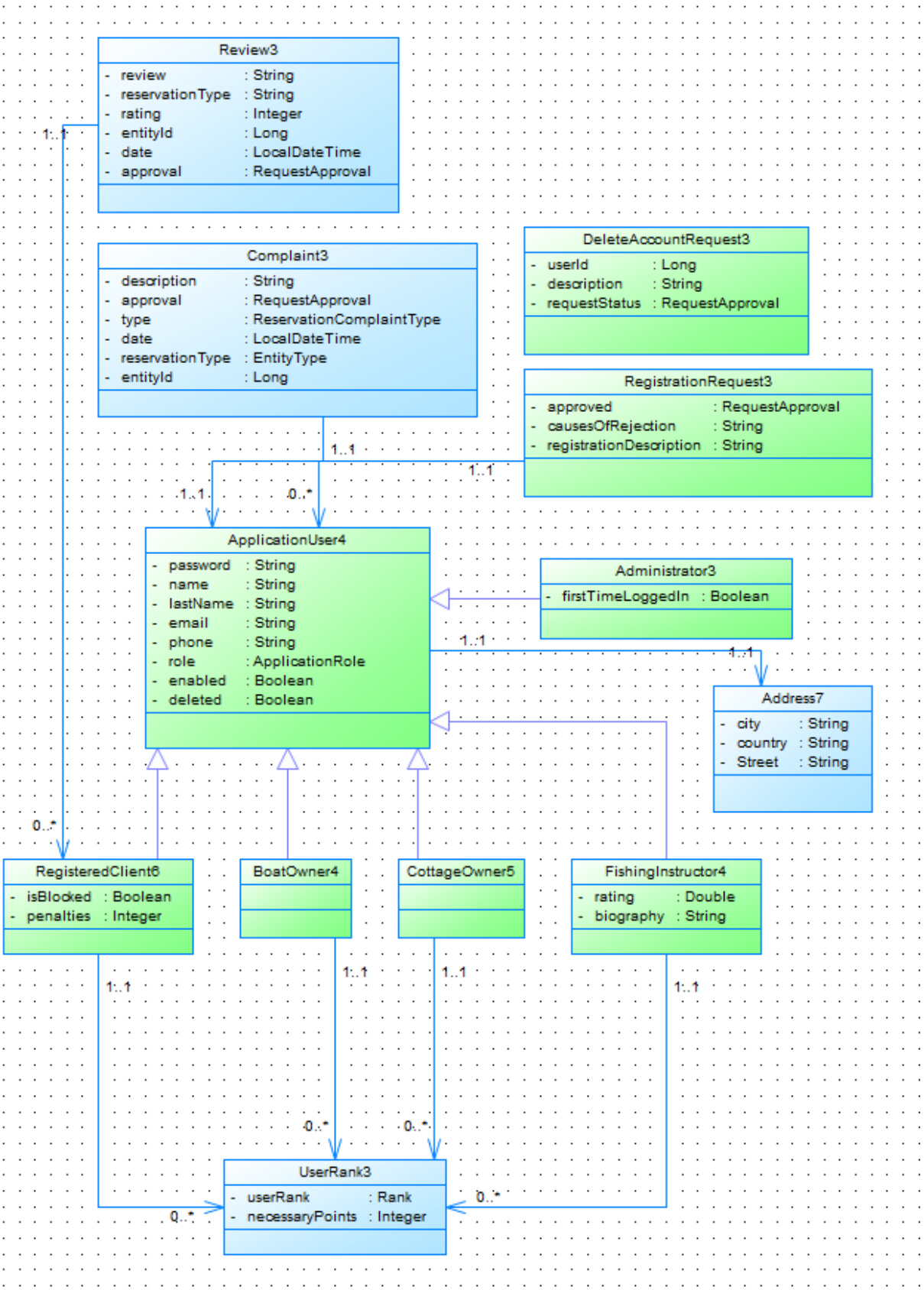
Glavni koncept organizacije klasa u okviru projekta jeste nasleđivanje univerzalne klase (**DatabaseEntity**), koja obezbeđuje automatsko generisanje ključa svih objekata baze.

Radi jednostavnije analize, u pratećem dokumentu dijagrami su prikazani po logičkim celinama.

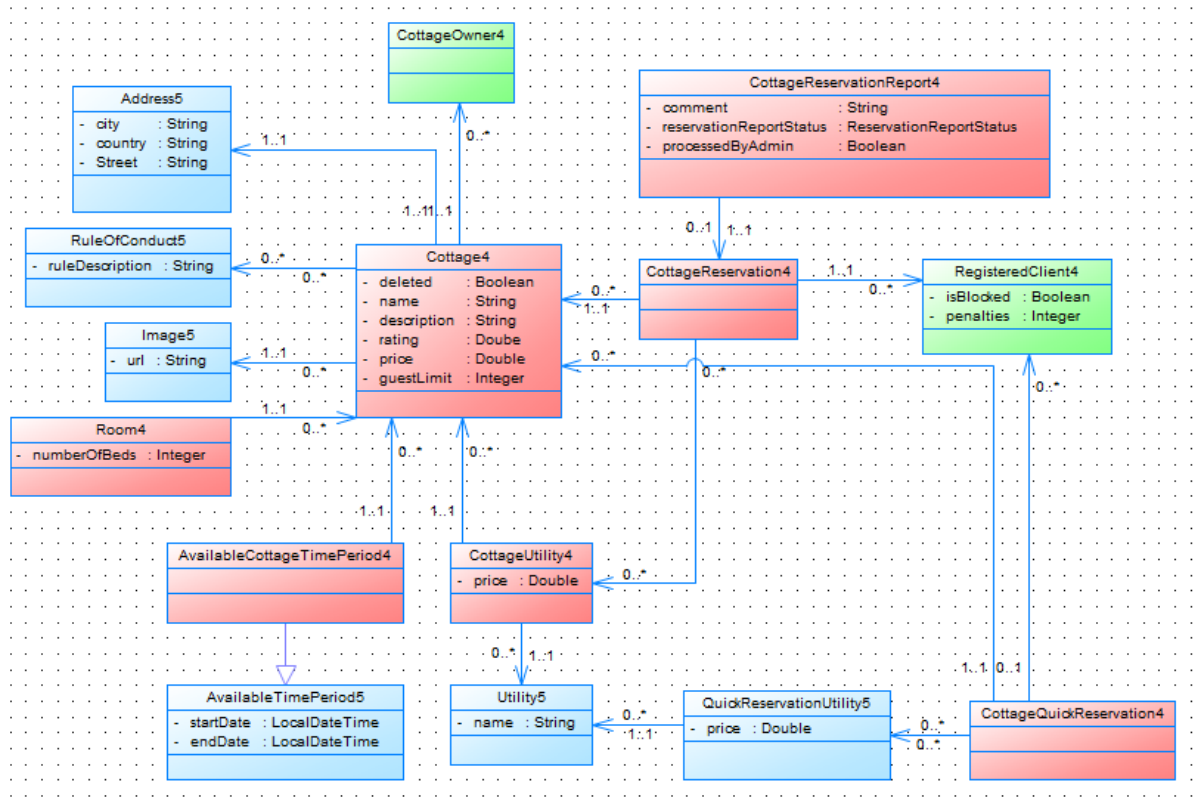
Klasni dijagram: Rezervacije i akcije (brze rezervacije)



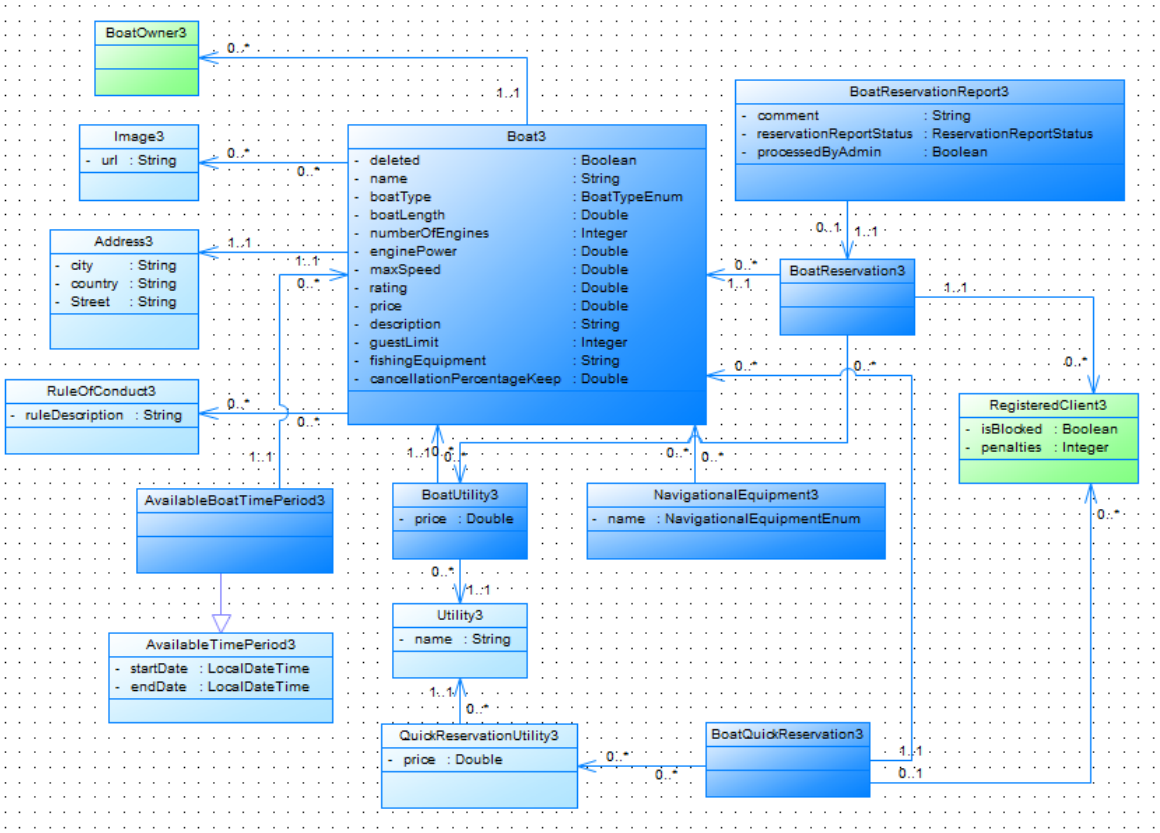
Klasni dijagram: Korisnici



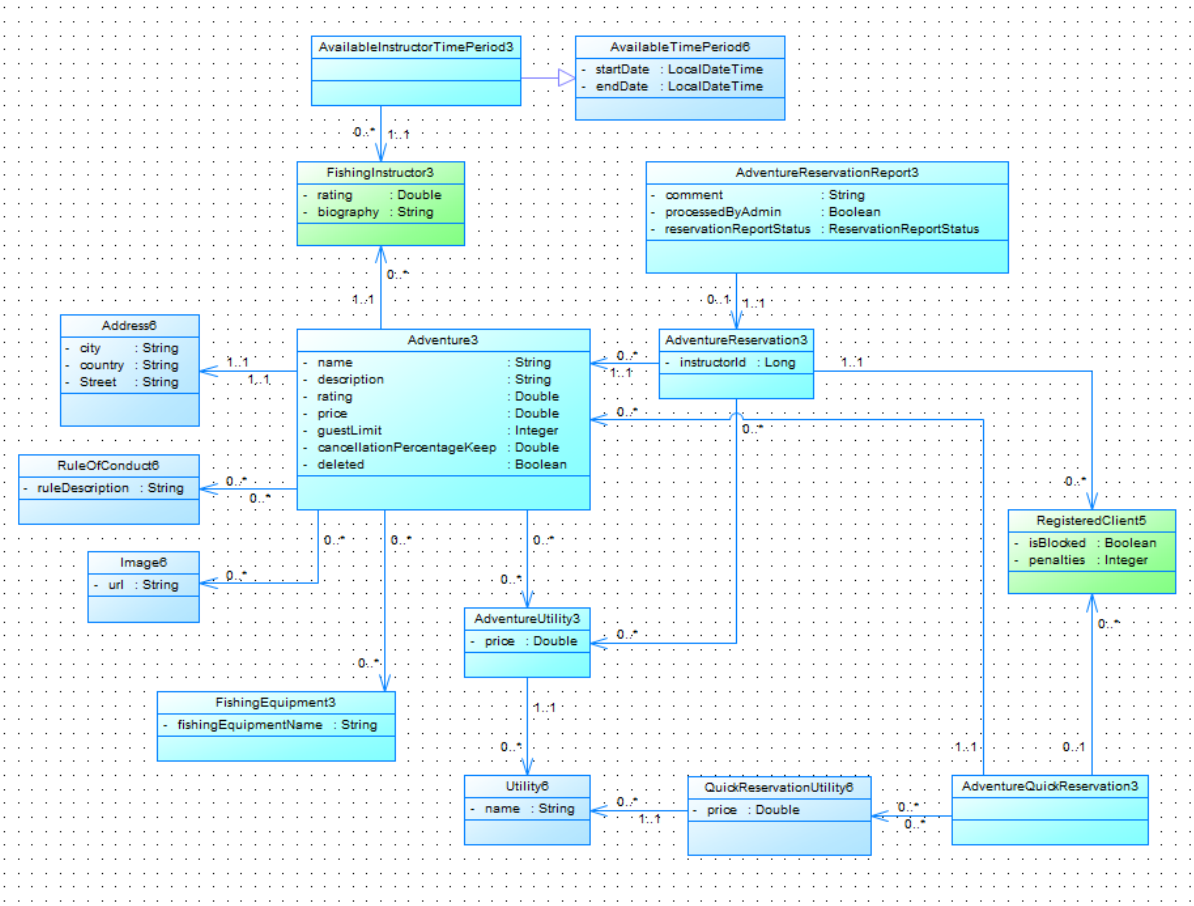
Klasni dijagram: Vikendice



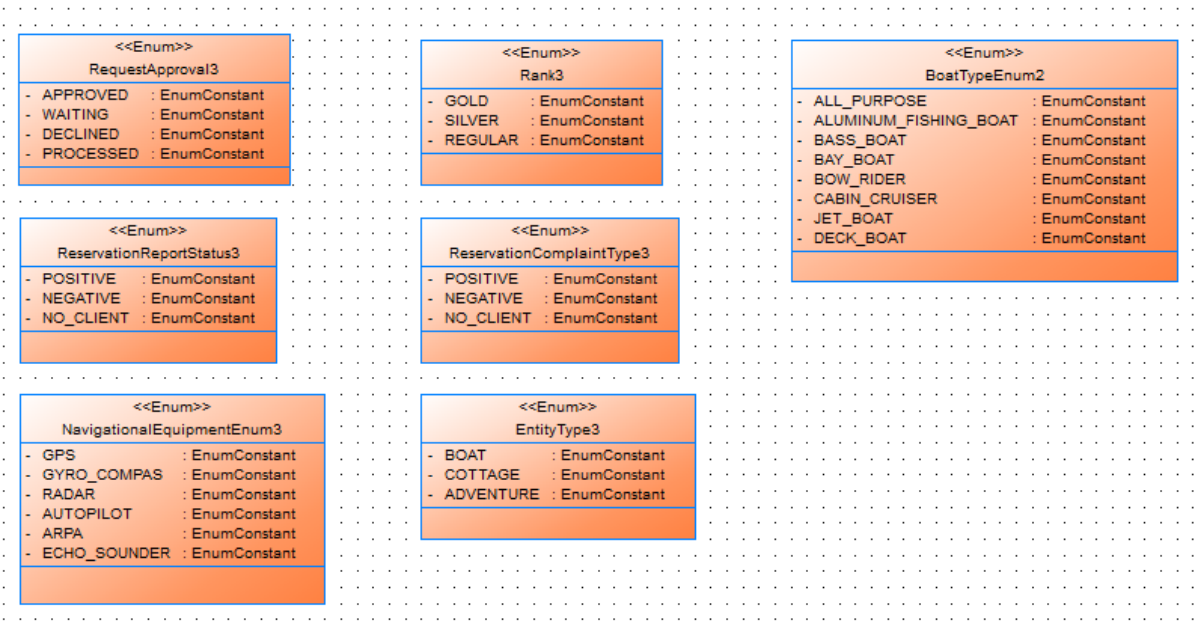
Klasni dijagram: Brodovi



Klasni dijagram: Avanture



Klasni dijagram: Enum



PREDLOG STRATEGIJE ZA PARTICIONISANJE PODATAKA

U velikim softverskim rješenjima podaci se dijele na particije kojima se nezavisno može pristupati i upravljati. Upotrebom particionisanja povećava se skalabilnost aplikacije i performanse.

Kada bi smo pokušali sve podatke da skladištimo na jednom serveru vrlo brzo bi došli do hardverskog limita. Zbog toga se vrši sklabilnost i trudimo se naše podatke da rasporedimo na više servera. Jedan od najčešće korišćenih pristupa jeste **vertikalno i horizonalno particionisanje podataka**. Tabele koje imaju veliki broj entiteta možemo horizonalnim particionisanjem podijeliti na više manjih po nekom ključu. U slučaju naše aplikacije to bi bile tabele sa rezervacijama. Tako bi mogli da podijelimo tabelu sa rezervacijama avantura na osnovu instruktora avanture. Id instruktora se tako možemo proslijediti hash funkciji koja će odrediti koji server baze podataka će čuvati rezervacije koje pripadaju određenom instruktoru. Ista situacija bi bila i sa Cottage Reservations i Boat Reservations. Takođe možemo izvršiti particionisanje rezervacija i po vremenu njihovog nastanka. Rezervacije koje su se desile u prošlosti za nas nemaju veliki značaj već se koriste za neke vrste statistike. Za njihovo smještanje bi koristitili hardver lošijeg kvaliteta. Takođe bi mogli da uradimo i vertikalno particionisanje rezervacija avanture. U jednoj particiji bi čuvali podatke kojima se rjeđe pristupa kao što su cijena avanture, ograničenje gostiju, id klijenta koji je izvršio rezervaciju...

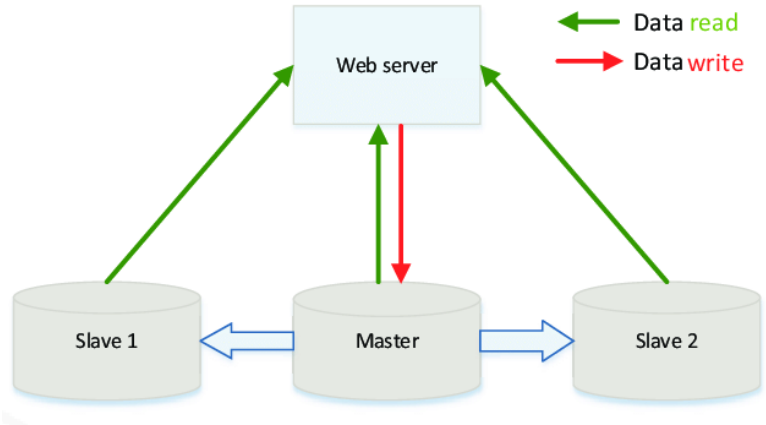
Dok bi u drugoj particiji čuvali početak i kraj rezervacije. Drugoj particiji će se češće pristupati zbog provjere da li se rezervacije preklapaju.

Particionisanje podataka takođe može u velikoj mjeri da poveća sigurnost naše aplikacije. Ideja bi bila da osjetljive podatke čuvamo na posebnim serverima kod kojih bi primjenili veće sigurnosne mjere. Tako možemo mailove,korisnička imena i lozinke korisnika čuvati na posebnim serverima. Na taj način bi se i veličina upita smanjila jer se ovi podaci rijetko koriste osim pri autehntifikaciji korisnika,dok drugim informacijama o korisnicima češće pristupamo.

PREDLOG STRATEGIJE ZA REPLIKACIJU BAZE I OBEZBEĐIVANJE OTPORNOSTI NA GREŠKE

Jedna od tehnika koja se koristi da bi se povećala brzina aplikacije jeste da podijelimo korisnike na osnovu njihovog geografskog položaja. Tako možemo postaviti mašine koji su blizi svakoj grupi korisnika (na svakom kontintentu možemo staviti po jedan server). To će u velikoj mjeri povećati brzinu pristupa podacima. Ove mašine bi bile međusobno povezane i radila bi se njihova sinhornizacija i na taj način bi korisnici u Evropi vidjeli iste podatke kao i oni u Americi.

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 zagušivala. Takođe ukoliko bi došli u situaciju da se master baza previše preoptrećenja zahtjevi 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 je očuvana *otpornost na greške*. Isto tako ukoliko dođe do neke prirodne katastrofe podaci će biti sačuvani jer imamo podatke na nasim slave bazama i pored toga na svakom kontintentu gdje se naša aplikacija koristi imamo mašine sa bazama podataka.



PREDLOG STRATEGIJE ZA KEŠIRANJE PODATAKA

Keširanje podataka je nezabolazna stavka prilikom visoko rastućih sistema. Velika količina podatka, kao i poziva na bazu mogu znatno da uspore sistem, a time i učine korisničko iskutvo manje prijatnim, odbijajući klijente od korišćenja veb aplikacije.

Ovaj proces najbolje je implementirati za **keširanje primarno statičkih podataka**, kao što su podaci o vikendicama, brodovima, avanturama, zatim komentari, i najznačajnije – slike. Ukoliko prepodstavimo da je na mesečnom nivou broj rezervacija milion, možemo videti da su podaci o dostupnosti entiteta visoko promenljivi. Često će se zakazivati nove rezervacije, te će se time dostupnosti entiteta rapidno menjati.

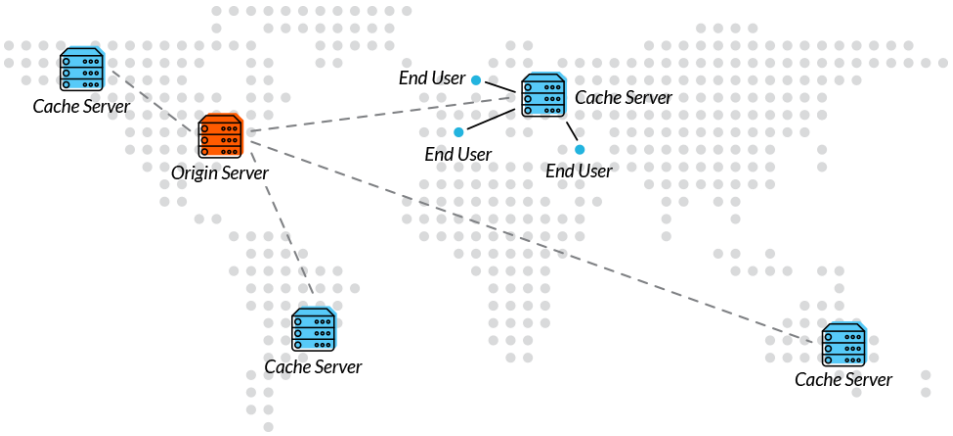
Ipak, i ove podatke možemo sačuvati u keš memoriji, s time što bismo ih menjali u kešu svaki put kada se oni ažuriraju u bazi.

Ovaj proces možemo da optimizujemo uklapajući ga sa **Event Sourcing**-om, opisanom u nastavku dokumenta. Za ograničenu keš memoriju, najpovoljnije je keširati podatke za koje znamo da se često dobavljaju: najpopularniji entiteti, periodi zauzetosti i dostupnosti istih u periodu relevantnom za trenutni datum. Pomoću analize pribeleženih događaja možemo izvući podatak o tome za koji period korisnici najčešće prave rezervacije. Na primer, ukoliko se ispostavi da većina klijenata rezerviše entietete mesec dana unapred, nema potrebe da se u kešu čuvaju podaci o dostupnosti za narednih 5 meseci.

Ukoliko se keš napuni, dobra strategija je ***LRU – Last Recently Used***, kojom se izbacuju najstariji dobavljani 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 slušaju 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.

Naša 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.



OKVIRNA PROCENA ZA HARDVERSKE RESURSE POTREBNE ZA SKLADIŠTENJE SVIH PODATAKA U NAREDNIH 5 GODINA

Pretpostavke sistema:

- Ukupan broj korisnika: 100 MILIONA
- Broj rezervacija svih entiteta na mesečnom nivou: MILION
- Sistem mora biti SKALABILAN I VISOKO DOSTUPAN
- Vlasnici usluga postavljaju prosečno 2 SLIKE po entitetu
- U proseku je 60% rezervacija rezultovalo i ocenom
- 5% korisnika sistema su vlasnici usluga, koji u proseku imaju po 2 ENTITETA
- 10% rezervacija su brze rezervacije

Zauzetost memorije [1] za skladištenje čestih pojedinačnih entiteta u bazi[2]:

- Adresa: 0.5 KB
- Application User: 2.7 KB
- Cottage: 2 KB
- Boat: 2 KB
- Adventure: 2 KB
- Image: ~600 KB
- Review: 1.4 KB
- Reservation: 0.6 KB
- Quick Reservation: 2.7 KB

Za 100 miliona korisnika potrebna memorija iznosi: **320 GB**.
Za procenjenih 40 000 000 entiteta potrebno je **6 010 GB** (računajući slike).
Za milion rezervacija mesečno, tokom godinu dana potrebno je **33 GB** za obične rezervacije i **16 GB** za brze rezervacije, ukupno **49 GB**.
Za procenjene ocene rezervacija tokom 5 godina biće potrebno **50 GB**.
Za skladištenje drugih entiteta slobodnom procenom izdvojićemo **20 GB**.

> Napomena: Sve procne rađene su za situacije koje bi zauzele više memorije, kako bi se izbeglo podcenjivanje potrebnih resursa.

UKUPNO, ZA ODRŽAVANJE SISTEMA U TRAJANJU OD 5 GODINA POTREBNO JE 6.5 TB.

[1] PostgreSQL Table Size
[2] Računanje veličine pojedinog entiteta

PREDLOG STRATEGIJE ZA POSTAVLJANJE LOAD BALANSERA

Ideja upotrebe load balancera jeste da opterećenje prenesemo na više servera i da jedan server ne obrađuje sve zahtjeve koje mu šalju klijenti. Stoga ćemo postaviti load balancer između klijenta i aplikativnog servera.

Prema nekim istraživanjima algoritam za load balancing koji daje dobre rezultate jeste „**Least Pending Requests**“. Ideja algoritma jeste da se prati broj zahtjeva koji određeni server ima pred sobom da obavi. Ukoliko je broj zahtjeva veći to će vrijeme za njihovo izvršenje biti veće. U realnom vremenu se prati broj zahtjeva koji svaki server treba da obradi i kandidat za novopristigli zahtjev je onaj server koji ima najkraći red zahtjeva. Algoritam može efikasno da se nosi sa kašnjenjem servera i vremenskim ograničenjima za zahtjeve. On automatski prepoznaje sve činioce koji će dovesti do produžetka redova zahjteva te nove zahtjeve prosleđuje na manje opterećene servere.

PREDLOG KOJE OPERACIJE KORISNIKA TREBA NADGLEDATI U CILJU POBOLJŠANJA SISTEMA

U cilju kreacije kvalitetne aplikacije glavni prioritet ne bi trebao da bude samo pravljenje tehnički ispravne i optimizovane aplikacije, već i one koja će svojim ponašanjem da reflektuje personalne potrebe svojih korisnika. U tu svrhu je **Event Sourcing** jedan od principa koje je potrebno primeniti na što širem spektru.

Kako su većina korisnika FishingBooker-a klijenti, a samim time i izvor profita kako za vlasnike usluga tako i za posrednu kompaniju, najveći fokus bi trebao biti na optimizovanju njihovog iskustva.

Pre svega, sistem **pretrage** i **pregleda stranica** posvećenih vikendicama, brodovima i avanturama bio bi značajan pokazatelj zainteresovanosti klijenta. Vreme provedeno na svakoj od pojedinih stranica bio bi efektivan faktor u stvaranju realne slike koliko su korisnici zainteresovani za pojedine entitete.

Aplikacija bi na osnovu ovih rezultata bila lako proširiva pokazivanjem „*Viewed most often*“ ili „*Recommendations*“ prikaza kako klijentima, tako i neautentifikovanim korisnicima. Dodatno, admini ili vlasnici usluga mogli bi da imaju pristup grafičkom prikazu koji bi na osnovu datih podataka priložio adekvatnu sliku o njihovim najuspešnijim i najtraženijim servisima, na osnovu kojih bi mogli adaptirati ostale usluge.

Dodatna akcija koja se prati mogla bi da bude prilikom samog **procesa zakazivanja**. Koliko često korisnici dođu do stranice pravljenja rezervacije i odustanu od iste? Koliko često korisnici otkažu prethodno napravljenu rezervaciju? Da li postoji korelacija sa cenom usluge, lokacijom, ili nedostatkom određenih informacija koji dovode do otkazivanja?

Za sistem je veoma bitan tok informacija između njenih korisnika. Primarno, ocene koje klijenti ostavljaju čine sistem bogatijim podacima, i lakše koristivim za nove korisnike. **Praćenje logovanja** na sitemu, kao i **ostavljanja komentara** za neocenjene rezervacije bio bi značajan u kontekstu boljeg podsticanja korisnika na datu aktivnost. Na primer, ukoliko je prošlo određeno vreme a korisnik se nije prijavio na sistem,

aplikacija bi mogla da pošalje mejl u kome ga obaveštava o najnovijim akcijama i novododatim entitetima koje je propustio. Slično, ukoliko se određena korisnička rezervacija završila a korisnik je nije ocenio, isti bi mogao da dobije notifikaciju koja bi ga periodično podsećala da bi to mogao da uradi. Na ovaj način bi se suptilno podigao promet FishingBooker-a, i korisnici bi idealno dali novim klijentima realnu sliku o uslugama koje su u ponudi.

Neke od 3rd party aplikacija koje mogu pomoći prilikom implementacije i analize događaja jesu **Dynatrace** i **SolarWinds Security Event Manager**.

KOMPLETAN CRTEŽ DIZAJNA PREDLOŽENE ARHITEKTURE