Univerzitet u Novom Sadu

Fakultet tehničkih nauka

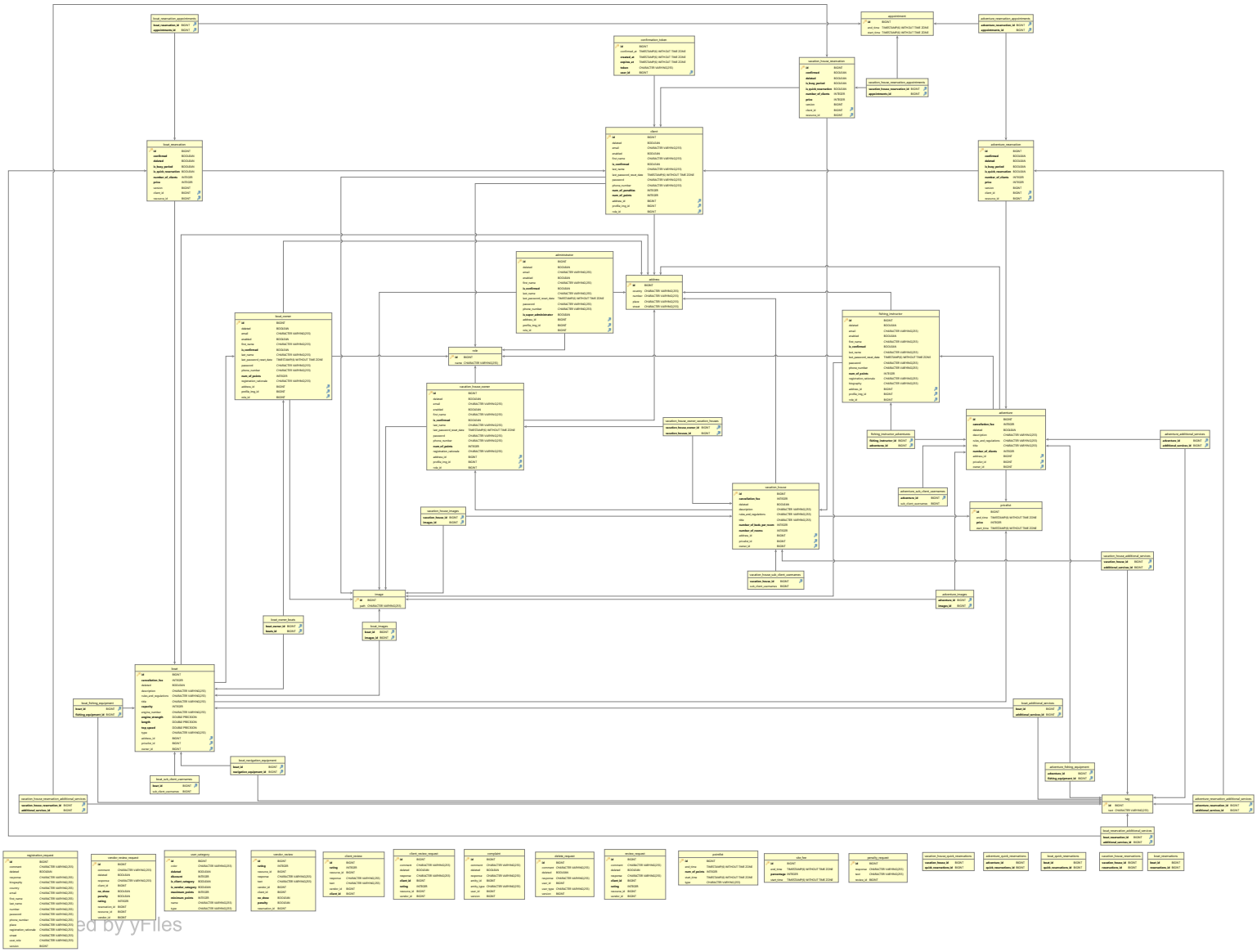
Skalabilnost

Tim 9 – Internet softverske arhitekture / Metodologija razvoja softvera

Studenti:

Miloš Milovanović SW 17/2019
Aleksandra Nedić SW 27/2019
Dunja Stojanov SW 30/2019

1. Dizajn šeme baze podataka



Slika 1 – Šema baze podataka

2. Predlog strategije za particionisanje podataka

Podaci kojima rukuje aplikacija nalaze se u bazi podataka i prilikom zahteva koji se obrađuje potrebno je pristupiti bazi. Kako pristup bazi može biti skup i vremenski dug, potrebno je odraditi particionisanje podataka kako vreme odgovora servera ne bi oduzimalo više od potrebnog.

Vertikalno particionisanje podataka trebalo bi se izvršiti prilikom prijavljivanja korisnika na sistem. Kako sam korisnik u bazi poseduje veliki broj atributa

(kolona), a za prijavljivanje su potrebni samo korisničko ime i lozinka, ove dve kolone se mogu izdvojiti u posebnu tabelu.

Horizontalno particionisanje podataka trebalo bi se odraditi nad tabelama rezervacija. Kako je rezervisanje najčešća operacija sistema i kako stalno pristižu nove rezervacije i korisnici gledaju istoriju svoji rezervacija, njihovo dobavljanje je potrebno optimizovati.

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

Obzirom da se naša aplikacija dosta služi prikazivanjem podataka, strategiju koju predlažemo da se implementira jeste master-slave tehnika replikacije baze. Ovom tehnikom ćemo omogućiti veliki broj operacija čitanja iz naše baze, samim tim korisnici će imate minimalno kašnjenje pri prikazivanu podataka. Sama tehnika se sastoji iz jedne master baze i više slave baza, gde će master baza podataka biti korišćena za operacije brisanja, ažuriranja i pisanja, dok će ostale baze biti korišćenje za operaciju čitanja. Takođe sve replike baze podataka će biti sihronizovane sa master bazom kako bi se održala konzistentnost podataka između svih baza. U slučaju pada master baze, kako sistem ne bi čekao njen oporavak, njena replika će biti zadužena za izvršavanje njenih operacija. U slučaju velikog broja operacije čitanja i pada jedne od baza replika, glavna baza privremeno može dobiti ulogu čitanja. Ovi mehanizmi obezbeđuju veću otpornost na greške kao i pregled podataka sa minimalnim čekanjem.

4. Predlog strategije za keširanje podataka

Pošto koristimo Hibernate, keširanja prvog reda je već podešeno na nivou sesije korisnika. Za keširanje drugog reda planiramo da koristimo EhCashe provajder. Tokom implementacije drugog reda keširanja želimo da postignemo što veći Cache hit ration kako bi naša aplikacija uvek pružala brz odziv na sve korisnikove zahteve.

Iskoristićemo keširanje unapred kao jedan vid povećanja skalabilnosti, gde ćemo keširati sve slike entiteta kao i pozadina aplikacije. Odlučili smo se za ovaj izbor jer su slike aplikacije velikog formata i zbog same osobine pretrage i filtriranja svih entiteta koje smatramo da će se najviše koristiti.

Keširanje po zahtevu klijenta ćemo koristiti pri prikazu svih informacija o entitetima. Kako je sama izmena entiteta jako retka operacija nećemo imati problem prikazom ovih informacija.

Prediktno keširanje unapred planiramo da iskoristimo po kreiranju samih entiteta i po praćenju preplaćenih entiteta klijenta. Ukoliko se napravi novi entitet na koji je korisnik pretplaćen, biće obavešten putem emaila, kako bi mogao da pogleda sve podatke izabranog entiteta.

Kako bi podaci u kešu bili sinhronizovani sa podacima u bazi, izabrali smo LRU strategiju izbacivanja kandidata iz keša. Kao posledicu izbora ove tehnike, korisnik će imati najbrži odziv operacijama koje najčešće koristi kako bi njegovo iskustvo aplikacije bilo što bolje.

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

Navešćemo okvirne procene hardverskih resursa za potrebe skladištenja nekih najbitnih eniteta, kako bi na kraju dali okvirnu procenu memorijskog zauzeća entiteta u narednih 5 godina:

1. Korsnici aplikacije zauzimaju približno 2.2kb po korisniku. Na 100 miliona korisnika to bi iznosilo 220gb
2. Resursi poput vikendice, avanture, broda približno zauzimaju 10kb po entitetu. Uz pretpostavku da u sistemu imamo 30% pružalaca usluga i da svaki pružalac usluga prosečno poseduje dva entitet, približno zauzeće memorije koje dobijamo je oko 580gb.
3. Rezervacije približno zauzimaju 0.6 kb. Uz pretpostavku da je broj rezervacija na mesečnom nivou milion i da svaka rezervacija ima jedan dodatan servis približno zauzeće memorije koje dobijamo je 12*5*1000000* 3.1kb(uz servis)=186gb
4. Recenzije zauzimaju približno zauzimaju 1.5 kb, spram toga ako imamo milion rezervacija na mesečnom nivou približno zauzeće memorije koje dobijamo je 12*5*1000000* 1.5kb=90 gb

Sumiranjem svih navedenih evaluacija dolazimo do toga da je potrebno 1076gb. Takođe u ovaj račun nismo uračunali ostale entitete, tako da bismo konačnu evaluaciju procenuli 2.5 do 3 puta većom. Konačna procena hardverskih resursa je oko 3tb.

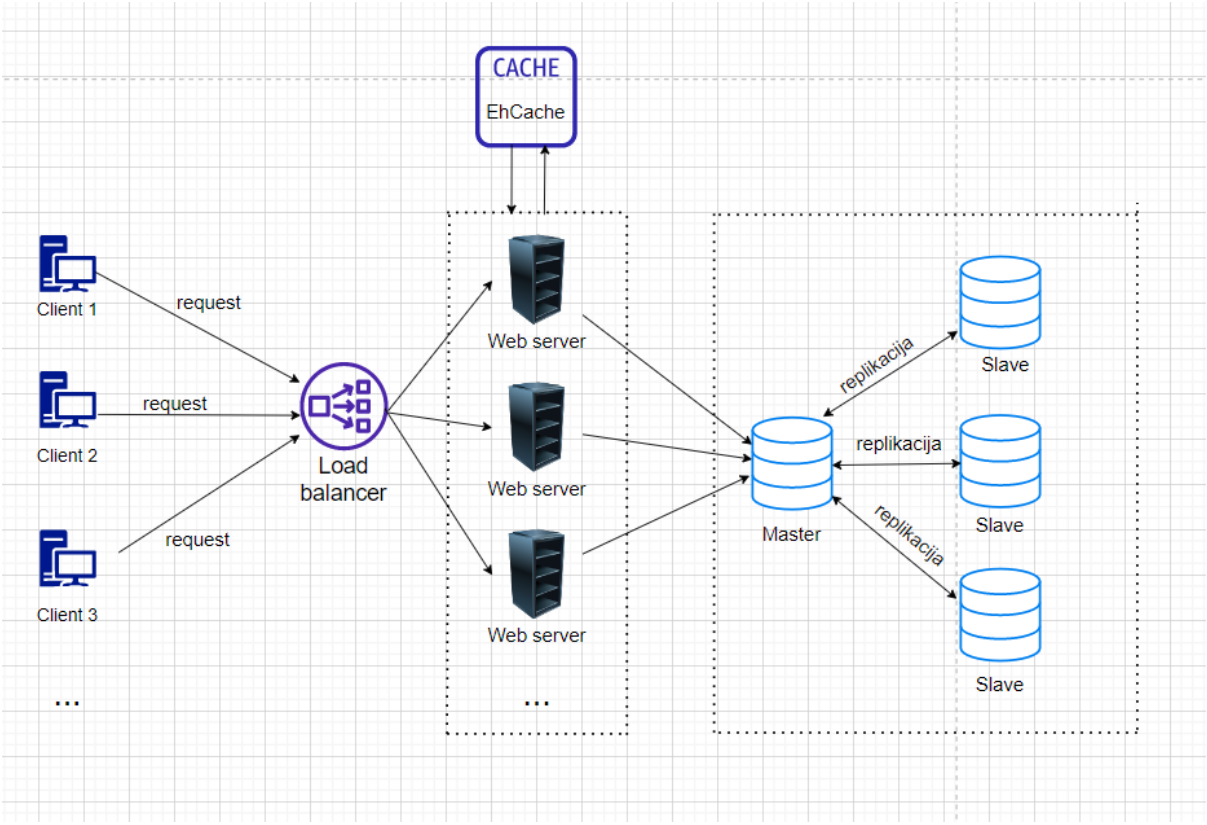
6. Predlog strategije za postavljanje load balansera

Kako našu aplikaciju korisni veliki broj korisnika dolazimo do zaključka da jedan server ne može opslužiti zahteve svih klijenata. Iz tog razloga potrebno je uvesti skaliranje. Vertikalno skaliranje predstavlja unapređenje servera u vidu povećanja memorije i procesorske moći. Ukoliko će korisnici vrlo retko koristiti neki server, vertikalno skaliranje bi se pokazalo pogodnim. Međutim, naša pretpostavka je da će veliki broj korisnika koristiti sve servere vrlo često jer sama aplikacija tako zahteva, iz tog razloga bi se implementiralo horizontalno skaliranje servera. Horizontalno skaliranje je takođe pogodno u slučaju otkazivanja nekog od servera, jer to ne bi izazvalo pad same aplikacije. Ispred servera postavio bi se Load balancer koji bi jedini imao javnu IP adresu, dok sami veb serveri nisu direktno povezani na internetu. Load balancer bi raspoređivao pristigle zahteve serverima po algoritmu Least connection. U ovom slučaju, load balancer bi samo vodio računa o opterećenjima pojedinačnih servera, ne vodeći računa o konkretnim HTTP sesijama korisnika, kako se one ne čuvaju na serveru, već u bazi podataka (stateless).

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

Operacije korisnika aplikacije je potrebno nadgledati u cilju poboljšanja i unapređivanja sistema. Kada govorimo o poboljšanju sistema, govorimo najčešće o vremenu odziva aplikacije. Kako svako korišćenje aplikacije podrazumeva pretragu i dobavljanje entiteta, ove operacije bi trebalo nadgledati. Nadgledanjem dobavljanja entiteta možemo saznati koje entitete korisnici najčešće zahtevaju i njihove podatke keširati kako bi se sam proces dobavljanja ubrzao. Takođe, kako naša aplikacija prvenstveno služi za rezervisanje, performanse ove operacije treba stalno analizirati kako bi user experince ostao pozitivan. Ukoliko se primeti da je rezervisanje sporo, možda je potrebno uvesti jos jedan server ili izvršiti optimizaciju samih algoritama.

8. Kompletan crtež dizajna predložene arhitekture (aplikativni serveri, serveri baza, serveri za keširanje itd)



Slika 2 – Dizajn predložene arhitekture