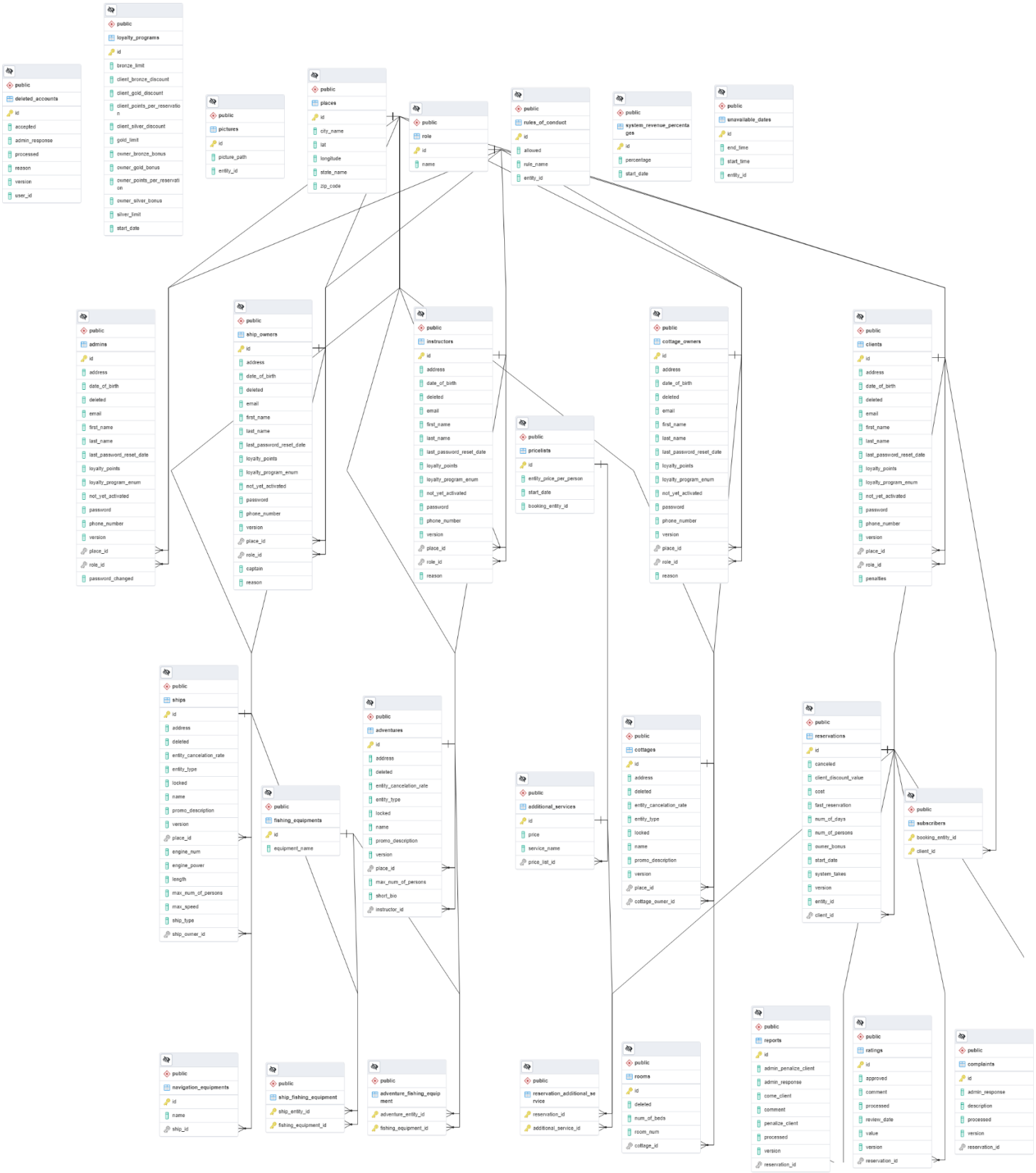
Skalabilnost ISA/MRS tim 5

školska 2021./2022. godina

Studenti
Mihal Sabadoš, SW20/2019
Zorica Vuković, SW21/2019
Rajko Zagorac, SW23/2019

1. Dizajn šeme baze podataka



Slika 1: Šema baze podataka

2. Predlog strategije za particionisanje baze podataka

Zbog predviđenog velikog broja korisnika aplikacije (100 miliona), količina podataka smeštenih u bazi bi bila velika. Ovo bi znatno usporavalo rad baze i celokupnog sistema i često bi dolazilo do preopterećenja zbog velikog broja operacija čitanja i pisanja. Predlog rešenja ovog problema bi bio korišćenje horizontalnog i vertikalnog particionisanja u ključnim delovima sistema kao što su tabele rezervacije i korisnika.

Tabelu rezervacije bi trebalo particionisati horizontalno zbog potrebe za svim podacima jedne instance i to na osnovu identifikatora entiteta. Svaki entitet ima popriličan broj svojih rezervacija i zbog toga bi mogli čuvati sve rezervacije jednog entiteta u posebnoj particiji čime bi rasteretili proces čitanja iz jedne tabele rezervacija i ubrzali rad sistema.

Pošto je predviđen veliki broj korisnika aplikacije i kako sistem čuva dosta informacija o svakom pojedinačnom korisniku, tabelu korisnika bi mogli vertikalno particionisati. Ovo bi uradili tako što se izdvoje podaci potrebni za autentifikaciju korisnika kao što su email i lozinka od ostalih podataka koji se koriste u ostalim funkcionalnostima sistema. Ovim bi izbacili nepotrebno dobavljanje podataka o korisniku prilikom same prijave, što bi doprinelo bržem i bezbednijem sistemu.

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

Pored particionisanja baze potrebno je dodatno optimizovati njen rad prilikom operacija čitanja podataka. Ovo možemo učiniti Master-Slave tehnikom replikacije baze kojom se dobija konzistentnost. Ovom tehnikom omogućavamo back-up baze koji bi bio od velike važnosti jer su podaci u ovoj aplikaciji bitni i često im se pristupa. Takođe bi sve replike bile sinhronizovane čime bi održali konzistentnost podataka između njih što omogućava korišćenje bilo koje replike u datom trenutku. U slučaju pada glavne baze, sistem ne bi čekao na njen oporavak već bi se prebacio na neku njenu repliku jer se u svakoj replici nalaze autentični podatke. Ovim se obezbeđuje veća otpornost sistema na greške kao i brži pristup bez čekanja.

4. Predlog strategije za keširanje podataka

Budući da koristimo Hibernate kao objektno relacioni mapper, keširanje prvog reda je već interno podržano na nivou sesije. Za keširanje drugog reda ćemo koristiti **EhCahce** provajder. Dok budemo implementirali keširanje, glavni cilj je što manji broj pristupa konkretnoj bazi podataka i što veći Cache hit ratio.

Prvo ćemo keširati statički sadržaj poput slika entiteta, i lokacije entiteta koji sadrže u sebi informacije o gradu i državi. Ovaj sadržaj ćemo keširati unapred, pre podnošenja zahteva od strane klijenta. Za ovaj pristup smo se opredelili prvenstveno zato što nam je cilj da klijenti u kratkom odzivu mogu da vide sve entiteta kako bi postigli bolji UX i veći broj rezervacija.

Kod prikaza svih entiteta zajedno sa svim informacijama, koristićemo keširanje na zahtev klijenta. Kako je pregled i pretraga svih entiteta češća operacija od same izmene entiteta, koja predstavlja relativno retku operaciju, smatramo da nećemo imati velikih problema sa zastarelom verzijom podataka. Ukoliko do ovog

problema dođe, rešićemo ga povremenim osvežavanjem podataka i konfigurisanjem dužine trajanja keša (*Time-to-live*).

Pored navedenih strategija koristićemo i prediktivno keširanje. Kako je prirodni tok akcija da nakon kreiranja nekog entiteta, rezervacije, žalbe… idemo na njegov prikaz ili izmenu, omogućićemo keširanje novokreiranog objekta kako bismo ubrzali vreme odziva. Imajući u vidu da imamo milion rezervacija na mesečnom nivou, i da recimo četvrtina klijenata nakon uspešne rezervacije ode u pregled iste, verujemo da bismo na ovaj način znatno smanjili broj pristupa bazi podataka.

Da bismo u kešu držali što više svežih podataka, moramo koristiti neku od tehnika izbacivanja podataka iz keša. Opredelili smo se za LRU strategiju (*Least Recently Used*) kao posledicu toga da se u našoj aplikaciji često pristupa upravo pretraživanim ili novokreiranim podacima.

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

Prikazaćemo okvirne vrednosti zauzeća podataka za najvažnije tabele u našoj aplikaciji, a na kraju ćemo dati procenu ukupnog zauzeća u narednih 5 godina.

- Korisnici -> oko 2kb po korisniku. Na 100 miliona korisnika ovaj broj iznosi **200gb**
- Entiteti -> oko 12kb po entitetu, uzimajući u obzir sve dodatne servise i opremu. Neka su 20% korisnika vlasnici, i recimo da svaki vlasnik ima samo jedan entitet, dobijamo 20 miliona entiteta. Ukupno iznosi oko **240gb**
- Rezervacije -> oko 0.5kb. Pretpostavimo da svaka rezervacija ima jedan dodatan servis, te joj to daje prosečnu veličinu od 3kb. Ukoliko imamo million rezervacija na mesečnom nivou dobijamo da će nam ukupno biti potrebno oko 12*5*1000000*3kb = **180gb**
- Žalbe -> recimo da imamo 25% posto žalbi na sve rezervacije, odnosno ukupan broj od 15 miliona. Kako jedna žalba zauzima oko 3.5kb, dobijamo ukupan iznos od oko 50gb. Zajedno sa odgovorom na žalbu ovaj broj može da poraste do **80gb**
- Izveštaj o završenim rezervacijama -> uzmimo da će svaka druga rezervacija imati svoj izveštaj i da je za skladištenje jednog izveštaja potrebno oko 2kb. Ukupno dobijamo 6*5*1000000*2kb = **60gb**

Na kraju, kada saberemo sve dolazimo do brojke od **760gb**. Naravno, ovde su uračunate samo najvažnije tabele, te ukoliko bismo posmatrali sve podatke, ovu brojku bismo mogli uvećati bar tri puta. Konačna procena hardverskih resursa iznosi oko **2280gb** ili oko **2.8tb**.

6. Predlog strategije za postavljanje load balansera

Kako je pretpostavka da će aplikacija biti korišćena od strane velikog broja korisnika, možemo da pretprostavimo da jedan server neće biti dovoljan da opsluži zahteve svih klijenata. Ovaj problem bi mogao da se reši upotrebom vertikalnog ili horizontalnog skaliranja. Pomoću vertikalnog skaliranja bi se kreirao server koji će imati veće hardverske mogućnosti nego dotadašnji server, ali bi ovaj pristup bio dosta skup i nekada nedovoljan, pa će se iz tog razloga koristiti horizontalno skaliranje aplikacije. To bi značilo da imamo više servera koji opslužuju korisnike, a postavljanjem Load Balansera koji ravnomerno raspoređuje zahteve ka serverima bi se znatno rasteretili serveri. Princip izabran za biranje servera kojem će zahtev biti poslat je Round Robin princip. On je izabran pre svega jer je vrlo jednostavan za implementaciju i pri tome daje odlične performanse, pri čemu se

vodi računa o trenutnom opterećenju samih servera. Pored toga, kako u našoj aplikaciji backend ne čuva podatke vezane za sesije korisnika (stateless je), možemo da zaključimo da nemamo ograničenje vezano za to da zahtevi jednog korisnika uvek moraju da pristižu na jedan server što dodatno olakšava posao LB.

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

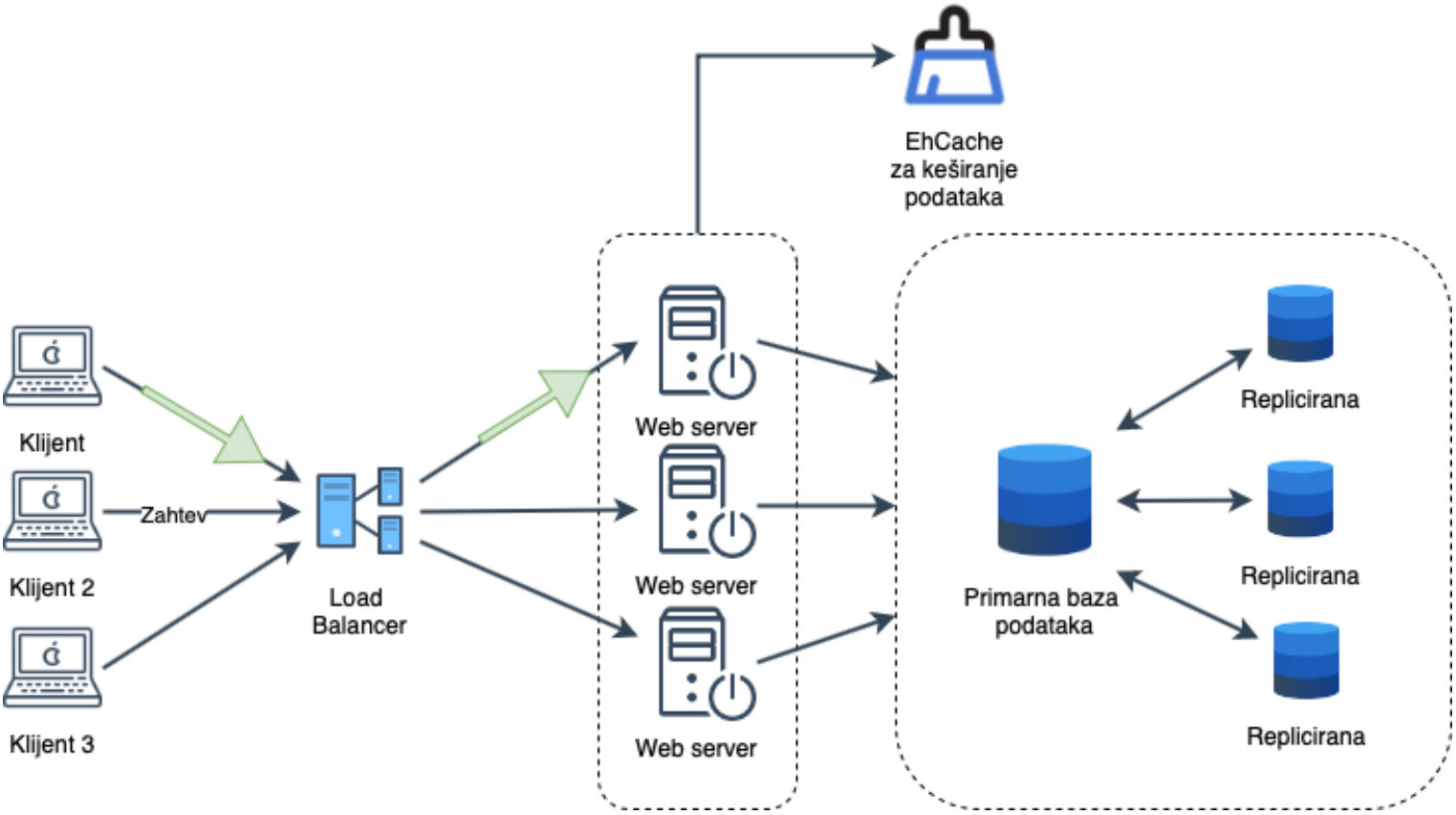
Za analizu celokupnog rada aplikacije bilo bi potrebno dosta vremena i resursa, što u većini slučajeva nije moguće, pa je neophodno u startu videti koje funkcije potencijalno mogu da dovedu do usporenja sistema. Kada se utvrdi koje to funkcije dovode do usporenja i uopšte prave određeni problem u aplikaciji, u većini slučajeva možemo lako da pronađemo uzrok i na kraju uspešno rešimo taj problem.

Sistemi za rezervisanje i izdavanje ponuda su danas jako popularni kod korisnika i zato je neophodno da oni budu što efikasniji. To bi značilo da bi bilo lepo da imamo mogućnost praćenja koje to entitete u sistemu korisnici gledaju kako bi mogli da formiramo neke pretpostavke o popularnim entitetima u sistemu i neke druge opcije koje bi korisniku olakšale samu pretragu. Kada smo kod usability-a bitno je nadgledati na koji način korisnici koriste aplikaciju i tako videti koliko im je sistem opterećujuć/lak za korišćenje. Pored toga, nadgledanjem potreba korisnika mogli bi da zaključimo kojim podacima se najčešće pristupa, koji podaci se najređe menjaju i na taj način omogućiti bolje keširanje u samom sistemu. Na ovaj način bi se slalo manje zahteva za čitanje ka bazi podataka, što bi svakako unapredilo rad sistema. Takođe, nadgledanjem korisnikovih potreba možemo da zaključimo koje su to najčešće korišćene operacije i ukoliko se utvrdi da njihovo vreme odziva nije idealno, možemo da se pozabavimo pitanjima asinhronog izvršavanja istih ili promenom algoritama unutar samih operacija. U našem sistemu operacije koje bi mogle da se nadgledaju su:

1. Pregled vikendica/brodova/avantura, kao i njihova pretraga i filtriranje
2. Rezervacija dostupnih entiteta
3. Prijava na sistem

Pored prethodno navedenih operacija za praćenje, veliki značaj treba dati i nadgledanju opterećenja servera. Ovo bi bilo od velike koristi ukoliko bi se utvrdilo da su neki serveri više opterećeni od drugih u sistemu i da postoji potencijalna mogućnost njihovog otkaza, jer bi onda bilo jasno da strategija koja je izabrana za ravnomerno raspoređivanje klijentskih zahteva nije adekvatna za našu aplikaciju i da je treba izmeniti.

8. Kompletan crtež dizajna predložene arhitekture



Slika 2: Predložen dijagram arhitekture