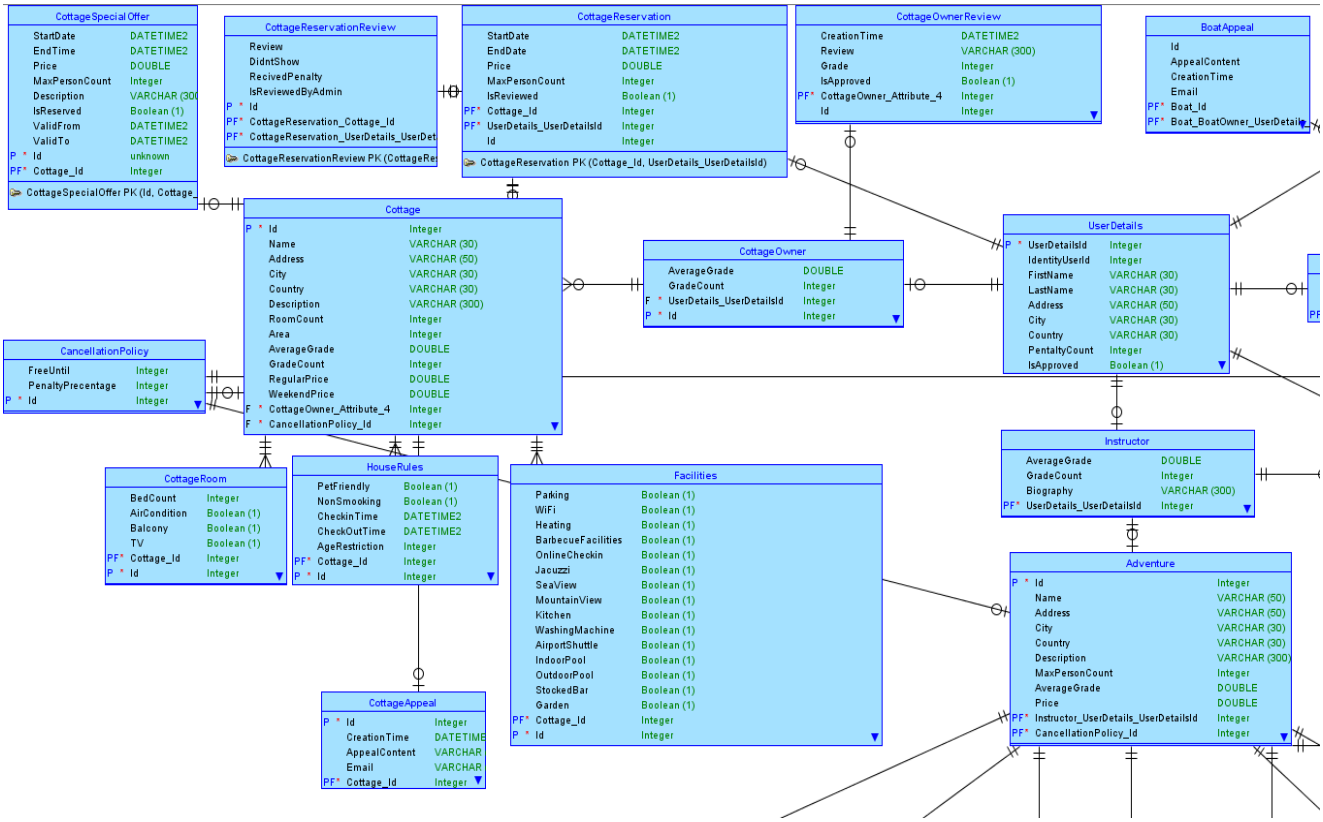
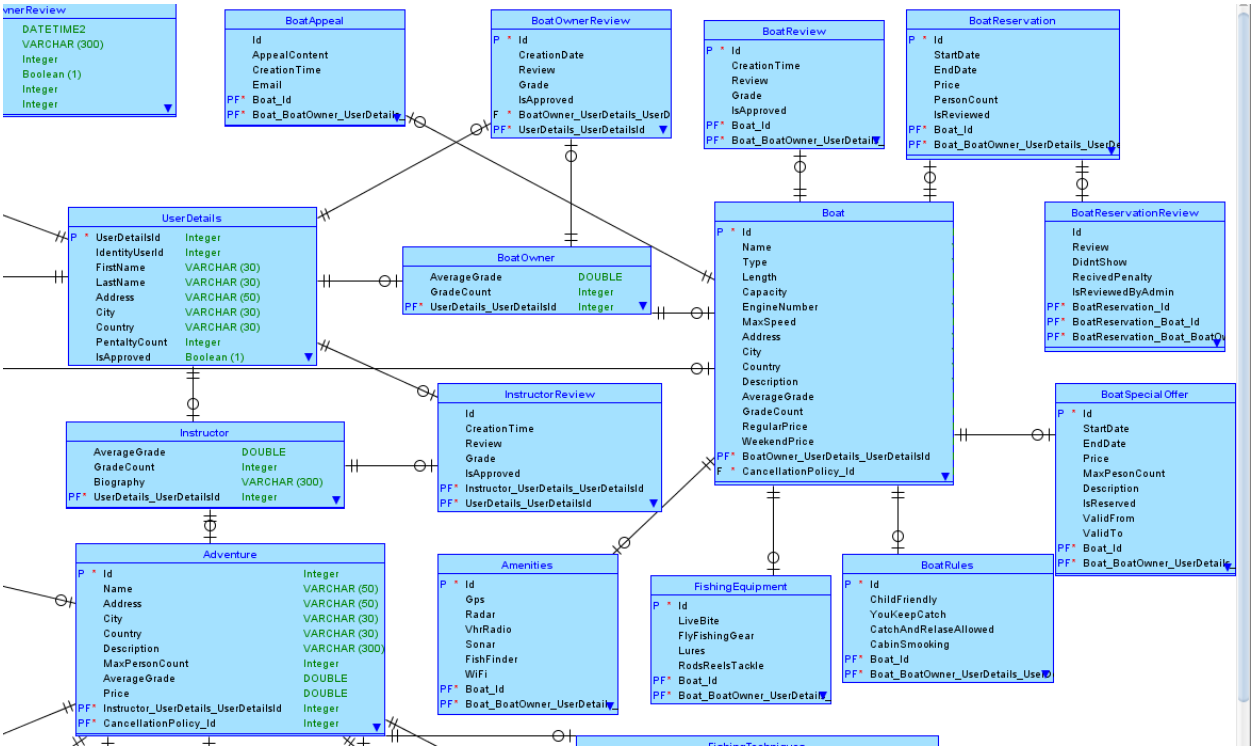
Skalabilna arhitektura web aplikacija

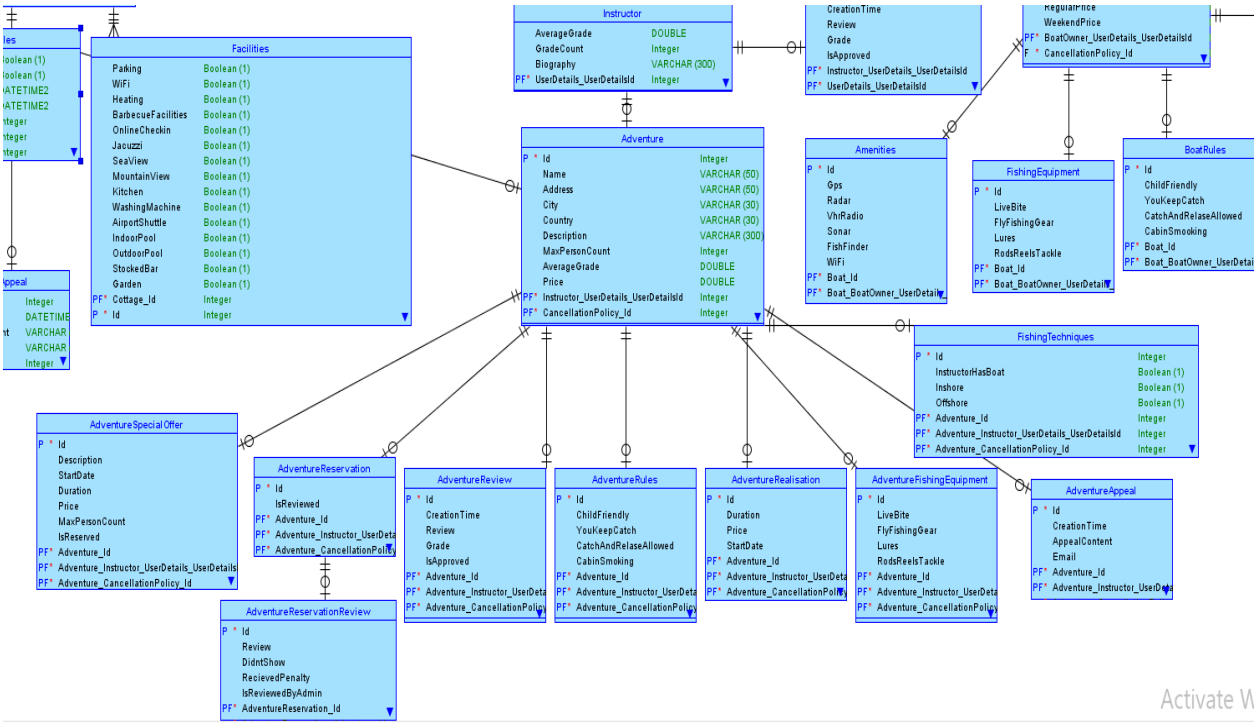
1. Dizajn šeme baze podataka



Slika 1 : Deo šeme baze podataka za vikendicu



Slika 2 : Deo šeme baze podataka za brod



Slika 3 : Deo šeme baze podataka za instruktora

2. Predlog strategije za particionisanje podataka

S obzirom na veličinu sistema i količinu saobraćaja, evidentno je da trenutni model baze neće moći da podrži toliki protok i pravovremeno odgovaranje na zahteve, pogotovo na zahteve za rezevacije entiteta. Prva ideja na koju dolazimo je svakako horizontalno particionisanje, zbog planiranog mesečnog saobraćaja od preko 200 miliona korisnika i količine podataka koja će konstantno rasti. Ukoliko dodamo i podataka da će se svakog meseca dodavati barem milion pregleda, horizontalna skalabilnost nam je neophodna. Zahvaljujući ovom pristupu, možemo iskoristiti i pogodnosti grupisanja podataka koji su čest odgovor na neki upit (SELECT/WHERE ili FIND), sa mogućnošću agilne promene plana raspodele podataka.

Pri prethodno opisanom obimu posla, moramo da obratimo pažnju i na nužno brisanje nepotrebnih podataka, što bi nam moglo biti olakšano ukoliko bismo napravili strategiju po kojoj bi oni bili čuvanu u istim delovima tabele. Za izradu projekta oslonili smo se na komplet proizvoda koji nudi kompanija Microsoft u okviru svog servisa Azure, te je korišćen SQL Server kao sistem za upravljanje relacionom bazom podataka. SQLServer na Azure-u nudi veliki broj pogodnosti za proširenje i skalabilnost, uz teoretski „neograničene“ resurse, koji naravno dolaze sa svojom cenom. Azure nudi opcije Sharding-a i multi-tenant i single-tenant organizacije, sa mogućnošću promene sistema, što nam daje fleksibilnost u budućnosti I na taj način možemo menjati našu bazu podataka s obzirom na obim korišćenja aplikacije.

Smišljanje strategije particionisanja podataka je veoma bitna stvar i veoma utiče na performanse aplikacije. Jedna od idelja je multi-level particionisanje je da rezervacije možemo podeliti u zavisnosti od id vlasnika vikendice/broda i instruktora, što je pogodan primer za podelu. Osim jedinstvene baze podataka, svaki od servera bi imao svoju bazu sa VIEW tabelama, kako bismo ubrzali protok i traženje informacija kroz aplikaciju – uz dodatak Azure Queue storage. Naravno, pogodno je razmotiri i ostale servise koji nude ovu uslugu (poput Kafke i RabbitMQ). Svi serveri bi bili povezani na AQS i bili pretplaćeni na informacije koje su im od značaja, kako bi što brže bili ažurirane njihove lokalne baze.

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

Za razliku od trenutne organizacije aplikacije, koja nije otporna na greške zbog samo jedne instance baze podataka i aplikacije na serveru, za aplikaciju ovih dimenzija ćemo morati da obezbedimo više kopija baze podataka (Slave-ova). Svaki put kada se izvrši uspešan commit na master (originalnoj) instanci baze, propagiraju se promene na sve replike. Na ovaj način u svakom trenutku obezbeđujemo otpornost na gubitak podataka ili prestanak rada jedne od instanci baza. Da bismo dodatno ubrzali proces rada naše aplikacije, sve upite koji neće vršiti commit-ove na bazu (read operacije) ćemo proslediti na „podaničke“ instance, a konzistentnost podataka nam neće biti ugrožena. Da bismo bili otporni na potencijalne greške, jednu od podaničkih instanci baze ćemo u svakom trenutku držati u potpunosti sinhornizovanu sa glavnom instancom slanjem logova komandi koje se izvršavaju nad originalnom instancom i na tu instancu. U slučaju pada glavne instance, komande će se izvršiti nad glavnom kopijom i korisnik neće moći da primeti razliku. Izabrali smo ovaj pristup (master – slave) zbog veličine aplikacije, koja iako velika, i dalje ne zahteva da bude geografski distrubirana i da ima master – master arhitekturu (gde bi svaka instanca imala mogućnost i write operacija), što bi ugrozilo konzistenciju. Azure nude sve ove opcije, te za njihovu implementaciju ne moramo da tražimo druge servise.

4. Predlog strategije za keširanje podataka

Keširanje podataka zarad ubzavanja rada aplikacije je planirano da se izvede na više različitih nivoa :

• Keširanje na nivou baze podataka – uz pomoć sistema Azure Redis Cache, koji je implementacija open-source Redis Cache sistema, omogućeno je smanjenje broja potrebnih sporih operacija na bazi uz pomoć keširanja, te se podaci dobijaju znatno brže.

• Keširanje na nivou servera – na jednom od servisa bismo postavili keširanje sesija korisnika koji su trenutno ulogovani, kao i njihov set permisija, kako bi gateway izbegao dodatni poziv na UserDetailsController. Takođe, uz pomoć planiranih View table smanjujemo broj neophodnih poziva na bazu podataka.

• Azure CDN – Azure Blob storage – uz pomoć ova dva sistema, uz pomoć proksi servera učinili bismo naše podatke pristupačnije korisniku, te bi se oni brže

učitavali. Poput verovatno najpozonatijeg Amazon S3, i Azure blob storage garantuje korisnicima pristup podacima i u slučaju globalne katastrofe, što donosi još jedan sloj sigurnosti za našu aplikaciju, Azure blob storage takođe koristimo u našoj aplikaciji za čuvanje multimedijalnih elemenata aplikacije.

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

Neophodni podaci za čuvanje u bazi podataka:

• Podaci o korisnicima: ukoliko je pretpostavlja da će našu aplikaciju ukupno da koristi 200 miliona korisnika i ako smatramo da se za svakog korisnika u proseku skladišti po 1000 bajtova, uz sačuvane promene podataka u prošlih 5 godina, okvirno zauzeće memorije iznosilo bi 400GB.

• Skladištenje podataka o vikendicama, brodovima i avanturama, procena je da nam neće trebati više od 5GB prostora za 5 godina. S obzirom na broj korisnika aplikacije i milion rezervacija mesečno, procena je da će u sistemu biti oko 2500 entiteta (vikendica, brodova i avantura) (400 pregleda u svakog entiteta), što nas dovodi do cifre od 12.5TB.

• Za skladištenje svih rezervacija, podatke o vlasnicima, korisnicima, instruktorima, ,specijalnim ponudama i dostupnosti uz procenu da po jednoj rezervaciji trebamo 1KB podataka, dolazimo do cifre od 20KB * 1.000.000 * 13 * 5 = 1.3TB.

• Za skladištenje svih ostalih informacija u sistemu (izveštaji o rezervacijama, recenzije, žalbe), za 5 godina je procena da će trebati oko 5TB.

Ukupna procena je da će našem sistemu biti potrebno oko 20TB. Na ovo treba dodati i Slave instance baze (gruba procena – barem 4) što daje 100TB, uz procenat podataka koji nam nisu potrebni oko 15-20%. Konačna računica nam daje 85TB za skladištenje svih podataka.

6. Predlog strategije za postavljanje load balansera

Plan arhitekture naše aplikacije postavlja gateway koji će da vrši pronalaženje servisa (service discovery) prilikom povezivanja na aplikaciju. Gateway bi predstavljao load balanser, koji bi se brinuo da zahtevi korisnika dođu do odgovarajućih servisa ili baze, koji bi ih preusmerio ukoliko neka od instanci određenog servisa nije aktivna ili je u kvaru ili su čak ugašene nepotrebne instance

servisa u periodima koju su prepoznati kao period manjeg saobraćaja na određenim servisima. Primer: noću bi vlasnici vikendica/brodova ili instruktori ređe koristili aplikaciju za kreiranje specijalnih ponuda, te bi load balanser mogao da ugasi neke instance servisa koji će vršiti ove operacije. Algoritam koji bi bio implementiran bio bi round-robin, u implementaciji Azure Load balansera.

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

Instaliranjem servisa Google Analytics i Azure Application Insights bismo mogli da ispratimo koje operacije korisnici najčešće koriste u našoj aplikaciji i mogućnosti daljeg poboljšanja servisa i keširanja. Na ovaj način možemo isprobanim sistemima da prikupimo dozvoljenu analitiku o ponašanju korisnika i pratimo da li će korisnici najviše koristiti pregled vikendica/brodova/instruktora/avantura, rezervacije entiteta, provere dostupnosti entiteta u nekom period, itd. Takođe, i način i vreme logovanja korisnika, kao i njihovi pokreti ili klikovi prilikom korišćenja aplikacije mogu nam ukazati da li je UI easy-to-use i user-friendly.

8. Kompletan crtež dizajna predložene arhitekture