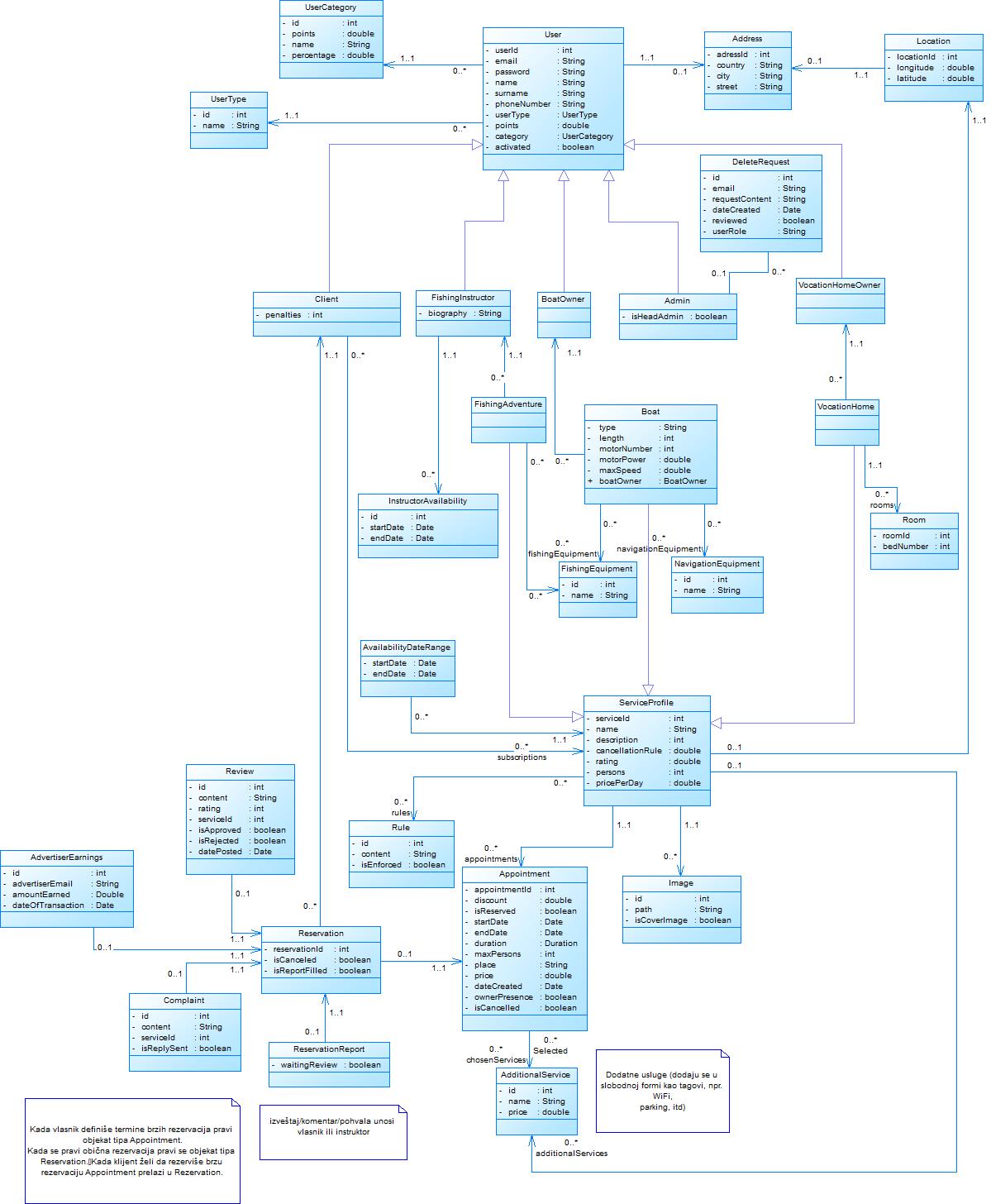
# Proof of Concept

TIM 18

## Dizajn šeme baze podataka



## Particionisanje podata

Prilikom započinjanja implementacije projekta, uočili smo tabelu koja potencijalno može da sadrži veoma velik broj podataka, gde bi se u većini slučajeva koristili samo neki delovi. Iz tog razloga, inicijalnu tabelu User-a smo vertikalno particionisali na više manjih tabela koji nam sadrže informacije neophodne za određenu vrstu korisnika. Na ovaj način rešavamo probleme čestog pristupa istoj tabeli radi izmene podataka koji ne moraju biti striktno povezani. Takođe, ukoliko razmatramo slučaj da naša aplikacija rukuje milionima entiteta, može se pojaviti slučaj gde će korisnici najčešće pristupati uslugama koje su veoma dobro rangirane, zatim uslugama koje su srednje rangirane, a najmanje će pristupati uslugama koje su najniže rangirane. Na ovom mestu bi izvršili horizonatalno particionisanje podataka i kreirali odvojene tabele. Ovako omogućavamo i da se tabela kojoj se najčešće pristupa potencijalno uskladišti „bliže“ klijentu kako bi mu lakše i brže bila dostupna.

## Strategije za replikaciju baze i obezbeđivanje otpornosti na greške

Kako imamo jednu instancu baze, veoma lako može doći do njenog otkaza. U tom slučaju čitava naša aplikacija nije funkcionalna te je neophodno izvršiti neka poboljšanja u arhitekturi našeg sistema. Ovaj problem je moguće rešiti uvođenjem nekoliko redudantnih instanci baze podataka i kreirati master-slave arhitekturu. U ovakvoj implementaciji preusmeravamo sve zahteve čitanja ka slave instancama, koji su mnogo češći, dok zahteve za pisanjem ostavljamo master instancama da obrađuju. Na ovaj način u velikoj meri rasterećujemo master baze jer su zahtevi za čitanje mnogo ređi. Takođe kako su master baze one koje upravljaju upisima, njih zadužujemo za očuvanje konzistentnosti, te smo rešili probleme koji mogu nastati u trenutnoj implementaciji. Međutim, i u ovakvoj implementaciji može doći do otkaza neke od baza. Ukoliko je u pitanju slave baza, master baza je zadužena da preuzme sav posao sve dok ne osposobimo slave bazu ili izmenimo trenutnu arhitekturu. Ukoliko dođe do slučaja da je jedna od baza koja je konfigurisana kao master padne, možemo unaprediti neku od postojećih slave baza da postane master, sve dok ne izvršimo potrebne izmene u arhitekturi. Na žalost, i dalje ostaje problem mogućeg gubitka podataka ukoliko master baza padne, jer naše slave baze ne moraju imati najnoviju verziju podataka, tj. poslednje izmene se nalaze isključivo na master bazama. U ovom slučaju rešenje bi bila izrada nekog recovery plana kako bi obezbedili očuvanje podataka.

## Strategija za keširanje

Kako bi dodatno smanjili pristup našim bazama i ubrzali proces komunikacije klijenta sa serverom, proširenje koje je moguće napraviti je implementacija keš memorije. Ukoliko pretpostavimo da naša aplikacija rukuje milionima entiteta, sigurno nije pogodno da sve to podatke šaljemo na svaki zahtev. To su podaci koji se u veoma maloj meri menjaju tokom vremena, te možemo iskoristiti neku od strategija keširanja i time takođe i smanjiti broj upita ka bazi.

Naša ideja je da se podaci keširaju na zahtev korisnika, tj. kada korisnik eksplicitno zatraži neki podataka. Na ovaj način će se u keš memoriji nalaziti najčešće pretraživani entiteti.

Takođe, kako je naša aplikacija implementirana koristeći javascript jezik, veliku količinu .js fajlova je potrebno učitati kako bi naša aplikacija bila prikazana. U ovome u velikoj meri pomaže CDN koji omogućuje keširanje naših statičkih fajlova na lokaciji koja je geografski najbliža klijentu i na taj način čini našu aplikaciju brzo dostupnu svim korisnicima širom sveta. CDN je podržan od većine web browser-a te dodatne konfiguracije i implementacija u nekim slučajevima može izostati.

## Okvirna procena za hardverske resurse potrebne za skladištenje svih

## podataka u narednih 5 godina

## Predlog strategije za postavljanje load balansera