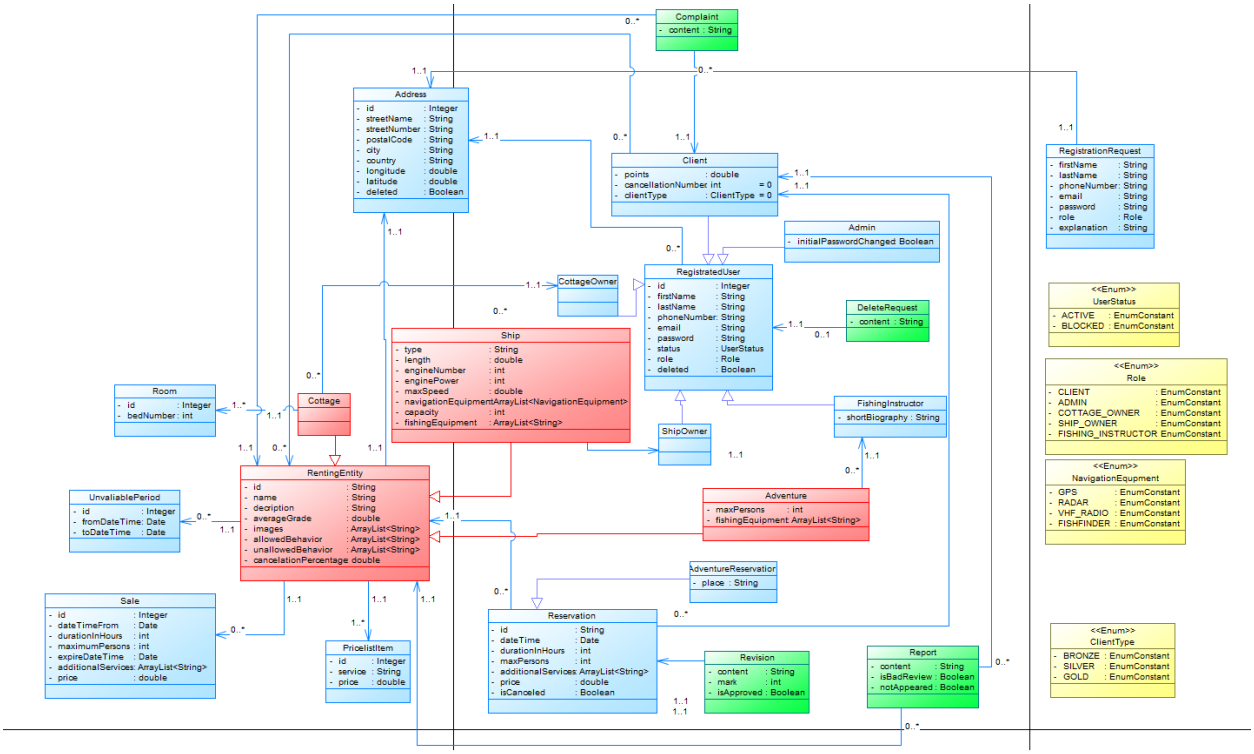
Skalabilnost

1. Dizajn šeme baze podataka (konceptualni)



Detaljan model se nalazi u fajlu ./ClassDiagram/ISA_Projekat.oom (rađeno u PowerDesigner-u).

2. Predlog strategije za particionisanje podataka

S obzirom da je glavna svrha ove aplikacije rezervacija različitih entiteta (vikendica, brodova i avantura), veliki je broj čitanja entiteta iz baze (kada klijenti razgledaju profile entiteta koje bi možda rezervisali), kao i pisanja odnosno izmena entiteta u bazi (prilikom rezervacije, dodavanja ili izmene entiteta od strane vlasnika). Radi bržeg čitanja i pisanja, potrebno je distribuirati podatke o entitetima na više mašina.

Predlog 1: Particionisanje na osnovu AdvertiserID-ja

- Prva ideja je da se raspodele podaci o oglašivačima a sa njima i o njihovim povezanim podacima (entitetima) na različite servere
- Kada nam je potreban entitet funkciji za heširanje se prosleđuje AdveriserID i onda će ona znati gde da traži entitet za čitanje, kao i gde ga treba sačuvati pri pisanju

- Mana ovog pristupa je što velika verovatnoća da će neki oglašivači imati veliki broj entiteta za izdavanje, dok će neki drugi imati svega 1 ili 2 entiteta, pa može doći do neravnomernog zauzeća memorije servera, čime bi neki serveri bili preopterećeni
- Mana ovog pristupa je što velika verovatnoća da će postojati popularni oglašivači, čiji se entiteti češće pregledaju i rezervišu od ostalih, pa su šanse velike da bi neki serveri bili opterećeniji od ostalih na kojima se nalaze zastareli ili entiteti lošijeg kvaliteta

Predlog 2: Particionisanje na osnovu RentingEntityID-ja

- Druga ideja je da se raspodele podaci o entitetima na različite servere
- Kada nam je potreban entitet funkciji za heširanje se prosleđuje RentingEntityID i onda će ona znati gde da traži entitet za čitanje, kao i gde ga treba sačuvati pri pisanju
- S obzirom da entiteti zauzimaju veću količinu podataka, ovakav pristup bi doveo do približno ravnomernog memorijskog zauzeća na svim serverima i rešio nedostatak prošlog pristupa
- Takođe bi bio rešen i problem popularnih oglašivača

U oba ova pristupa, javlja se jedan problem: korisnici sistema bi najčešće pretraživali entitete, gde bi iz liste svih entiteta određenog tipa nalazili ono što ih zanima. Pri traženju svih entiteta određenog tipa iz baze, moralo bi se proći kroz svaki server da bi se prikupili ti podaci jer ne znamo na kojem je serveru koji tip entiteta. Zatim bi neki centralni server sve te podatke objedinio i poslao odgovor. Ovakav pristup može prilično loše uticati na performanse.

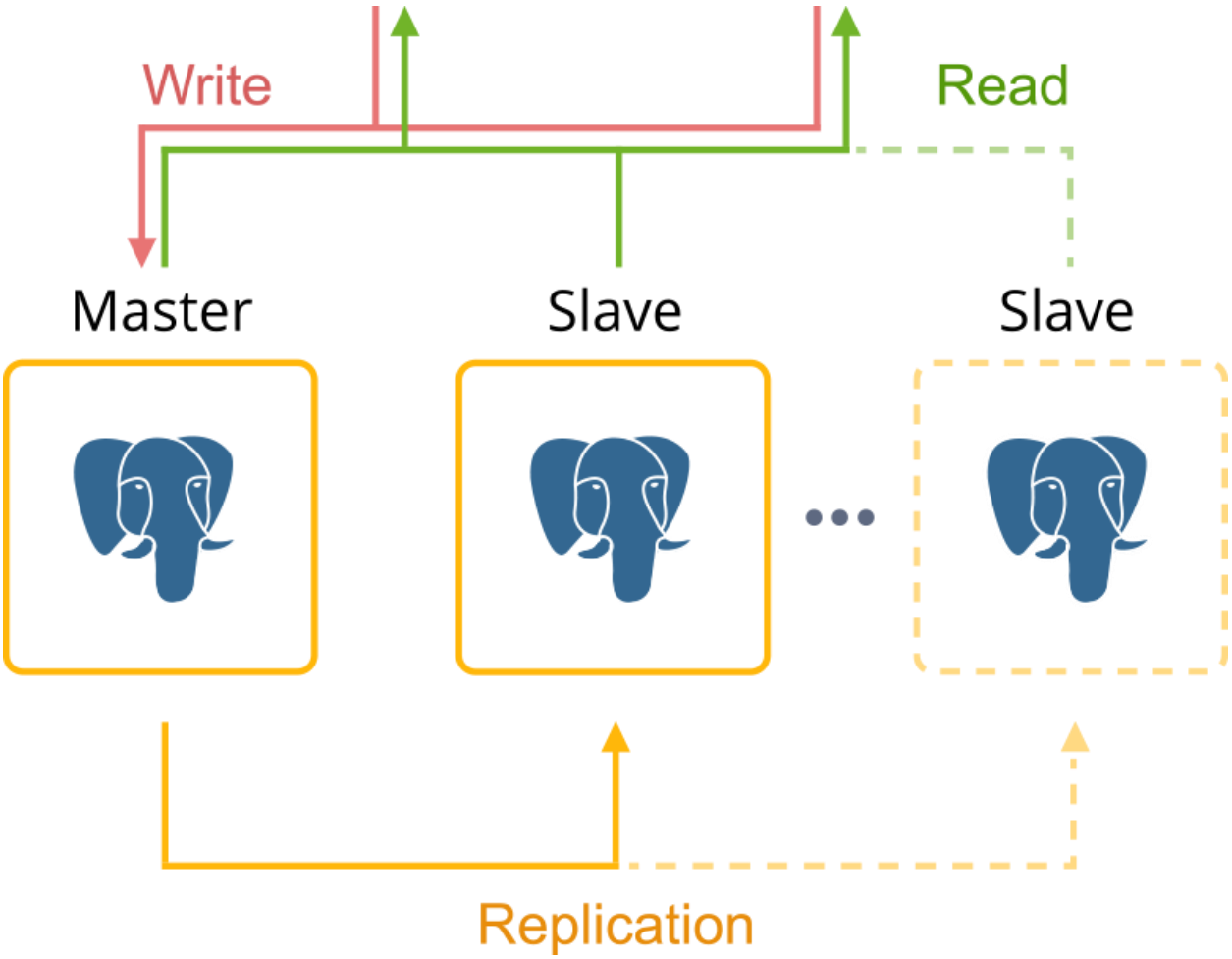
Predlog 3: Particionisanje na osnovu tipa entiteta i RentingEntityID-ja

- Ako bi broj servera bio deljiv sa 3 (za 3 tipa entiteta), onda bismo mogli rezervisati određene servere za određeni tip entiteta, a onda bismo u okviru njih particionisali podatke na osnovu RentingEntityID-ja
- Ovaj pristup bi rešio prethodno pomenuti problem prikupljanja svih entiteta određenog tipa, a zatim i doneo sve benefite koje donosi predlog 2.

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

Sistem koji smo implementirali se pretežno zasniva na čitanju podataka (dobavljanje entiteta, istorije rezervacija, aktuelnih rezervacija, akcija, dostupnih perioda i sličnog). Stoga, prirodno je da imamo više sekundarnih (*Slave*) servera koji će služiti samo za čitanje podataka, a upis novih podataka može obavljati primarni (*Master*) server baze podataka.

U slučaju da primarni server nije dostupan, iz bilo kog razloga, možemo odraditi *failover*, odnosno odabrati neki sekundarni server, koji će postati novi primarni.



4. Predlog strategije za keširanje podataka

Primer keširanja podataka implementiran je u kodu. Korišćenjem Ehcache-a, implementirano je keširanje svakog od entiteta (vikendice, broad i avanture). Konfiguracioni fajl ehcache.xml se nalazi u folderu ./FishingBooker/backend/src/main/resources.

```
<cache alias="cottage" uses-template="default">
    <key-type>java.lang.Integer</key-type>
    <value-type>com.backend.model.Cottage</value-type>
    <resources>
        <heap>2</heap>
    </resources>
</cache>
<cache alias="ship" uses-template="default">
    <key-type>java.lang.Integer</key-type>
    <value-type>com.backend.model.Ship</value-type>
    <resources>
        <heap>2</heap>
    </resources>
</cache>
<cache alias="adventure" uses-template="default">
    <key-type>java.lang.Integer</key-type>
    <value-type>com.backend.model.Adventure</value-type>
    <resources>
        <heap>2</heap>
    </resources>
</cache>
```

U odgovarajućem servisu za svaki entitet (*CottageService.java*, *ShipService.java*, *AdventureService.java*), anotacijama *@Cachable*, *@CachePut* i *@CacheEvict* (u *EntityService.java* jer se brisanje entiteta radi na nivou nadklase) je naznačeno šta treba keširati ili čitati iz keša. Primer iz klase *AdventureService.java* i *EntityService.java*:

```
@Cacheable("adventure")
public Adventure findById(Integer id) { return adventureRepository.findById(id).get(); }
```

```
@CachePut(cacheNames = "adventure", key = "#adventure.id")
public Adventure update(Adventure adventure) throws IOException {
    Adventure adventureToUpdate = this.findById(adventure.getId());
    adventureToUpdate.setName(adventure.getName());
```

```
@Transactional
@CacheEvict(cacheNames = "entity", key = "#id")
public void deleteEntity(Integer id) {
    if(isEntityBooked(id)) throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "Entity is now booked.");
```

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

Pretpostavke:

- Ukupan broj korisnika aplikacije je 100 miliona
- Broj rezervacija svih entiteta na mjesečnom nivou iznosi milion
- Sistem mora biti skalabilan i visoko dostupan
- 7% sistema čine vlasnici entiteta, koji u prosjeku imaju po 2 entiteta
- 5% rezervcija su akcije (brze rezervacije)
- Entiteti u prosjeku imaju po 5 pravila dozvoljenog i nedozvoljenog ponašanja, i 10 stavki u cjenovniku
- Vikendice u prosjeku imaju 3 sobe
- Navigaciona oprema u prosjeku ima 2 stavke
- Oprema za pecanje u prosjeku ima 3 stavke
- Entitet u prosjeku ima 1 period nedostupnosti
- Entitet u prosjeku ima 1 definisanu akciju
- Svaki string u prosjeku sadrži 12 UTF-8 karaktera (12B)

Memorijsko zauzeće najzastupljenijih entiteta u sistemu (po torci):

- Address: 82B

- RegisteredUser: 78B = 74B + 4B (AddressId)
- PricelistItem: 28B = 24B + 4B (EntityId)
- UnavailablePeriod: 15B
- RentingEntity: 176B = 164B + 4B (AdressId) + 4B (UnavailablePeriodId) + 4B (SaleId)
- Room: 12B = 8B + 4B (CottageId)
- Cottage: 44B = 36B (3 sobe) + 4B (RentingEntityId) + 4B (CottageOwnerId)
- NavigationEquipment: 12B = 8B + 4B (RentingEntityId)
- FishingEquipment: 4B
- Ship: 92B = 40B + 4B (RentingEntityId) + 4B (ShipOwnerId) + 3*FishingEquipment + 2 *NavigationEquipment = 40 + 4 + 4 + 36 + 8
- Adventure: 32B = 12B + 4B (RentingEntityId) + 4B (InstructorId) + 3*FishingEquipment
- Reservation: 84B
- Sale: 84B
- Image: ~500KB

Neophodni hardverski resursi za rezervacija na mjesečnom nivou: 84B * 10^6 ~ 80MB

Neophodni resursi za čuvanje entiteta: 7m vlasnika/instruktora * 2 entiteta * (176 + 1/3 * (44 + 92 + 32)B + 2 slike * 500KB + 1 adresa * 82B + 1 vlasnik * 78B) ~ 13TB

Neophodni resursi za čuvanje korisnika (klijenti i administratori) : 90m * (78B + 82B adresa) = 15GB

Za period od 5 godina: 13TB + 15GB + 80 * 12 * 5MB ~ 13TB

Procena za propusni opseg

- Pretpostavka je da ima ≃ 33340 rezervacija po danu.
- Tri puta više pregleda entiteta ≃ 100000 po danu.
- Svaki entitet ima prosečno dve slike po entitetu (prosečno 500KB po slici).
- (100000 * 500kB) / 86400s slika ≃ 0.578 MB/s
- (33340 * 84B) /86400s rezervacija ≃ 32.4 B/s
- (100000 * 176B) / 86400s entitet ≃ 0.2 KB/s
- Ukupno: ≃ 0.579 MB/s ili 50 GB dnevno.

6. Predlog strategije za postavljanje Load Balancer-a

- Postoje razne strategije i algroitmi za održavanje klijentskih zahteveva širom serverskih grupa.
- Least connection algoritam predstavlja dinamički algoritam koji u trenutno pristigle zahteve raspoređuje na 5 najmanje aktivnih servera u u trenutku pristizanja zahteva.

- Weight least connection algoritam predstavlja nadogradnju na least connection algoritam koji još dodaje i karakteristike servera u algoritam prilikom raspodele zahteva.
- Admin dodeljuje težine serverima u zavisnosti od kapaciteta upravljanja saobraćajem i u zavisnosti od trenutno aktivnih konekcija servera i serverske težine raspoređuje korisničke zahteve na servere.

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

Nove prijedloge za poboljšanje sistema možemo dobiti i njegovim aktivnim posmatranjem. Prikupljanjem relevantnih podataka, dobijamo uvid u to kako sistem radi i uočavamo potencijalne nedostatke, koji nas vode do novih rješenja za unaprijeđivanje.

U slučaju našeg sistema, od značaja su podaci o broju rezervacija po određenom entitetu/avanturi, na osnovu čega utvrđujemo koji entiteti/avanture su traženiji pa primjenjujemo odgovarajuće strategije keširanja za „popularnije“ entitete.

Takođe, od značaja je i koje entitete klijenti najviše posjećuju u okviru sajta, pa se na osnovu toga mogu unaprijediti sistemi za preporuke za određene entitete i akcije vezane za njih.

Pored toga, mogu se pratiti i periodi kada klijenti vrše najveći broj rezervacija (oko praznika, neradnih dana, u koje doba dana/godine) i na osnovu toga dobiti informacije kada je najveća posjećenost, odnosno saobraćaj aplikacije.

8. Kompletan crtež dizajna predložene arhitekture