# Skaliranje

## 1. Dizajn šeme baze podataka

- Konceptualni dizajn
- Logički dizajn
- Fizički dizajn

Sistem koji je već impelmentiran ima sređene veze između klasa. Korišten je spring boot zarad rukovanja entitetima. Korišten je JPA interfejs za postavljanje upita ka bazi, a kao baza korištena je PostgresDB. Svaka tabela ima polje id koji predstavlja ključ. To je jedinstvena vrijednost koja se automatski generiše prilikom pravljenja ojekta klase. Većina upita je sinhrona, dok su neki upiti radjeni asinhrono, kao što je dobavljanje perioda ažuriranja. Zarda povećanja performansi, u repozitorijumima su postavljani *custom* upii ka baz, koji za cilj imaju dobavljanje nekih polja, a ne svih.

## 2. Strategija za particionisanje podataka

Strategija particionisanja podataka igra ključnu ulogu u optimizaciji performansi, upravljanju velikim količinama podataka i skalabilnosti sistema.

Pošto je na početku potrebno obezbjediti resurse koji će zadovoljiti sve zahtjeve korisnika, najbolje je koristiti vertikalno skaliranje. Odnosno, obezbjediti hardver sa dovoljno procesorske moći i sa dovoljno memorije. Nakon toga nakon što se pveća broj korisnika mogu se dodati novi serveri. U slučaju da se broj korisnika počne povećavati, ovo nike dovoljno, pa je potrebno razmisliti i strategiji particionisanja monolitne u mikroservisnu arhitekturu. Tu bi imali mikroservis za autentifikaciju i autorizaciju, mikroservis za upravljanje korisničkim nalozima, mikroservis za upravljanje pregledima i mikroservis za nabavku/isporuku klijentima. U zavisnosti od daljeg razvoja treba razmisliti da je bolje imati svaku odvojenu bazu za svai mikroservis ili dijeljeu azu. U svakom slučaju najbolje je da imamo jednu bazu u servisu za autentifikaciju i autorizaciju, pošto je ovo jedan od najkoristenijih servisa.

## 3. Strategija za replikaciju i otpornost na grešaka

Pošto je potrebno osigurati da svi servisi budu visko dostupni, na nivou ove arhitekture je najlakse replicirati service na više servera. U slučaju otkaza jednog, nastavit ce sa radom drugi serveri, iako ce trpiti povećano opterećenje. Ovo iziskuje dosta više materijalnih resursa. U slučaju kada bi se implementirala mikroservisna arhitektura, bilo bi jednostavnije izanalizirati ranjive service, koji često trpe otkaze i za njih obezbjediti rezervne resurse.

## 4. Strategija za keširanje podataka

U trenutnom sistemu implementirano je keširanje na nivou *spring boot* frejvorka. Servisi koji  često rade sa istim podatcima podrzavaju keširanje. Trenutno je podržano keširanje sa maksimalno 100 instanci. Smmatramo da je ovaj broj dovoljan, da smanji opterećenje ka bazi, a u isto vrijeme da obezbjedi konzistentne podatke. U slučaju da se iskoristi svih 100 mjesta, brišu se najstariji ili najmanje korišteni objekti.

U slučaju potrebe može se implementirati *Redis* baza podataka koja će služiti za keširnje veće količine podataka. Ovo je pogodno, jer se očekuje povećanje broja korisnika, a samim tim i količina podataka i količina upita k bazi.

## 5. Procena hardverskih resursa

Prije nego sto se sistem smjesti na neku hardversku platformu najbolej, je implementirati neko početno riješenje. Nakon toga potrebno je izanalizirati trenutno stanje sa podacima, brojem zahtjevai slično. Nakon toga potreno je odrediti strategiju skalianja, da li se radi vertikalno ili horizontalno skaliranje.

## 6. Strategija za postavljanje load balansera

U pocetnim fazama razvoja nema potrebe za load balanserima. Nakon što se uvedu nove instance servera, javiće se potreba za loud balanserom. Jedna od prvih koraka je da se izabere pogodan loud balanser i da se konfiguriše. On prije svega treba biti robusan, pošto mora da podrži sve zahjeve koji stižu k aplikaciji. Takođe, treba razmotriti, da li će load balanseri slati zahtjeve ka svim serverima, ili će postojati rezervni serberi koji se aktiviraju kada jedan od akivnik otkaže.

## 7. Nadgledanje korisničkih operacija

Samom upotrebom spring boot frejmvorka urađen je ispis logova. U slučaju pojave neke greške ispisaće se ta informacija u konzoli. U metodama je implementiran ispis poruka u slučajevima kada se desi neki izuzetak ili kada je neka aktivnost uspješno odrađena. Nedostaje trajno čuvanje logova, jer bi se u slučju kraha servera izgubila inormacija zbog čega ej to došli. Zarad bolje analize performansi, praćeja opterećenja i daljih strategija optimizacije sistema najbolje uvesti sistem monitrigakoji bi pratio korisnčke zahtjeve, upotrebu resursa i sve što to odrazumjeva

## 8. Zaključak

U radu je predstavljen prijedlog strategije za prvi period rada, kao i mogući prijedlozi koji se mogu koristii u slučaju povećanja broja korisnika. Dalju strategiju je na početku teško odrediti, moguće je raditi na osnovu prethodnig iskustva ili iskustva drugih. Nakon određenog vremena, kada se upotreba sistema poveća, najboje je raditi analizu zahtjeva, potreba,trenutnih performansi i na osnovu toga implementirati neku od navedenih startegija.