Proof of Concept

- ## Šema baze podataka

  U repozitorijumu, Klasni dijagram.pdf.

- ## Particionisanje podataka

  Kako je pretpostavka da postoji veliki broj korisnika, pregleda, lekova i apoteka, potrebno je distriburiati podatke na više mašina, kako bi se čitanje i pisanje odvijalo efikasnije, uvođenjem horizontalnog i vertikalnog particionisanja. Za korisnike bismo primenili vertikalno particionisanje, gde bismo u jednu tabelu izdvojili lične podatke, a u drugu ostatak informacija. Za tabelu Appointment predlažemo horizontalno particionisanje po datumu, na ovaj način zahtevi za dobavljanje pregleda bliski sadašnjosti će zahtevati prolazak kroz manji broj podataka jer očekujemo da najviše zahteva za dobavljanje pregleda bude za preglede koji su bliski sadašnjosti. Za tabelu MedicalReport predlažemo particionisanje analogno particionisanju za tabelu Appointment jer se za svaki Appointment vezuje jedan MedicalReport. Za tabelu Order takođe bismo primenili horizontalno particionisanje po datumu, kako bismo lakše dobavili trenutno aktivne porudžbine. Tabelu Pharmacy bismo vertikalno particionisali, radi lakšeg pristupanja stalno potrebnim podacima, podelili bismo to na dve tabele, gde bi prva sadržala osnovne podatke o apoteci (adresa, naziv, ...) dok bi druga sadržala sve preglede, lekove, cenovnik i zaposlene.

- ## Replikacija baze i obezbeđivanje otpornosti na greške

  Za replikaciju baze podataka mogli bismo da koristimo *master*/*slave* metod. Ovaj metod sadrži jedan glavni server kom se šalju podaci, koji dalje te podatke prosleđuje ostalim serverima (*slave*). Međutim zbog mogućnosti prestanka sa radom glavnog servera potrebno je da podaci budu sinhronizovani. Kako bi se sistem rasteretio, mogli bismo iskoristiti *slave* servere za čitanje, dok bismo *master* koristili za upis podataka.

- ## Keširanje podataka

  Keširanje podataka se uvodi kako bi pristup bazi podataka bio brži. Koristeći keš dodatno bismo poboljšali performanse aplikacije, samim tim rasteretili bismo opterećenje nad bazom podataka. CDN bi mogao da se koristi u ovoj stuaciji. Podaci koje bismo keširali su:

  1) apoteke – podaci o apotekama su nam uvek potrebni, pacijenti preko profila apoteke rezervišu preglede kao i lekove

2) pregledi – prilikom rezervacija pacijentima je potreban prikaz predefinisanih termina, pa je samim tim velika potreba za njihovim dobavljanjem
3) lekovi – prilikom rezervacija je potreban prikaz dostupnih lekova, kao i njihovih cena, iz toga razloga potrebno ih je keširati

Kako bismo balansirali između opterećenja baze i ažuriranosti podataka u kešu, konfigurisali bismo *time to live*.

- **Procena za hardverske resurse potrebne za skladištenje svih podataka u narednih 5 godina**

  Ako je broj rezervacija lekova i zakazanih pregleda kod farmaceuta i dermatologa na mesečnom nivou 1 000 000, i ako to u proseku zauzima 17KB u bazi, dobijamo da za 5 godina

  1 000 000 * 12 * 5 * 17 = 1 020 000 000KB

  i ako imamo 200 000 000 korisnika koji u proseku zauzimaju 8KB u bazi, dobijamo

  8 * 200 000 000 = 1 600 000 000KB

  što ukupno daje 2.5TB.

- **Postavljanje load balansera**

  Raspoređivanje opterećenja na više serverskih računara može da dovede do toga da naš sistem može da obradi više zahteva u jedinici vremena. Algoritam koji bismo mi iskoristili je *round robin,* koji podrazumeva da se zahtevi serverima šalju sekvencijalno.

- **Operacije koje treba nadgledati u cilju poboljšanja sistema**

  Kako bismo odredili performanse naše aplikacije, i poboljšali performanse sistema, posmatrali bismo broj pristiglih zahteva za registraciju pacijenata na dnevnom nivou , broj rezervacija lekova i pregleda u satu i danu, broj porudžbina za lekove na dnevnom nivou.

- **Crtež dizajna predložene arhitekture**