Proof of Concept (PoC)

Jelena Popov i Nikola Brodić – Tim 16

Za razvoj aplikacije koristili smo *Spring Boot framework*, baziran na *Java* programskom jeziku, na *backend*-u i *Angular 8 framework*, baziran na *TypeScript* programskom jeziku, na *frontend*-u. Obzirom da je *Spring Boot framework* baziran na *Java* programskom jeziku koji podržava *multithreading*, jedna pokrenuta instanca aplikacije može istovremeno opsluživati više korisnika.

To se postiže dodeljivanjem zasebne niti svakom pristiglom klijentskom zahtevu. Pošto je kreiranje niti dugotrajan proces, vrši se tzv. *Thread pooling* kojim se unapred kreira određen broj niti koje čekaju klijentske zahteve. U cilju podrške većeg broja korisnika, broj unapred kreiranih niti iz *pool*-a se može povećavati, ali je problem što smo hardverski ograničeni brojem procesorskih jezgara na kojima se paralelno mogu izvršavati aktivne niti.

Rešenje za navedeni problem omogućavaju vertikalno i horizontalno skaliranje. Vertikalno skaliranje podrazumeva unapređene postojećeg servera dodavanjem resursa kao što su procesor, radna memorija i disk. Horizontalno skaliranje, sa druge strane, podrazumeva podelu sistema na više manjih servera koji obavljaju istu funkciju i uvođenja *load balancer*-a. Veliki broj korisnika sa sobom nosi i veću verovatnoću otkaza sistema, što bi u slučaju vertikalnog skaliranja značilo nefunkcionalnost celokupnog sistema. Zbog toga bi se mi odlučili za horizontalno skaliranje, kod kog otkaz dela sistema ne utiče na rad ostatka, tako da korisnici ne bi ni primetili da je do otkaza došlo. Ovakav pristup se može primeniti na našu aplikaciju, jer je komunikacija sa klijentom potpuno *stateless*, pa server ne čuva informacije o sesiji.

Postoje različite strategije kojima *load balancer* raspoređuje klijentske zahteve između servera. Mi bismo se odlučili za *Least Response Time* strategiju koja, osim što uzima u obzir broj aktivnih konekcija, uzima i vreme potrebno serveru da klijentu vrati odgovor. Na ovaj način se obezbeđuje da klijent što pre dobije odgovor na poslati zahtev. Radi veće bezbednosti, možemo dodati i još jedan redudantni *slave load balancer* koji se aktivira u slučaju da prvi otkaže i tako smanjuje *down time* aplikacije.

Replikacija baze podataka je proces skladištenja istih podataka na više lokacija radi poboljšanja dostupnosti i pristupačnosti podataka, poboljšanja otpornosti i pouzdanosti sistema i poboljšanja performansi. *PostgreSQL* baza podataka koju smo upotrebili u našem projektu u tu svrhu, između ostalog, nudi *master-slave* režim rada. *Master* baza podataka predstavlja glavnu bazu, u koju se vrši upis i iz koje se čitaju podaci. *Slave* baza podataka predstavlja *backup* bazu koja se nalazi na *standby* serveru. Sve transakcije koje se obavljaju na *master* bazi su uspešno *commit*-ovane tek ako su uspešno zapisane i u *master* i u *slave* bazu – tzv. *2-safe replication*. To znači da je sadržaj

slave baze podataka uvek sinhronizovan sa *master* bazom. Iz tog razloga, ona se može koristiti za čitanje podataka da bi se rasteretio saobraćaj sa glavnom bazom, a može automatski postati i *master* baza u slučaju da server *master* baze otkaže. Moguće je imati i više *slave* baza podataka koje prihvataju zahteve za čitanje podataka, pa da *master* baza samo prihvata upise, što predstavlja *single leader* replikaciju. Upravo je to strategija replikacije za koju bismo se odlučili u našem sistemu, jer zbog velikog broja korisnika postoji i velika potreba za upisom, a još više čitanjem podataka. Prethodno opisani način replikacije baze podataka se može specificirati za upotrebljenu *PostgreSQL* bazu putem konfiguracionih fajlova.

Particionisanje baze podataka podrazumeva podelu nekih tabela na manje delove koji predstavljaju particije, kako bi se poboljšale performanse. Upiti se lakše i brže sprovode nad manjim celinama u odnosu na cele tabele koje sadrže mnogo veći broj torki. Moguće tehnike particionisanja kod *PostgreSQL* baze podataka su *range* i *list partitioning*, kao i njihove kombinacije. Pošto je broj zakazanih pregleda i operacija na mesečnom nivou veoma velik, mi bismo primenili *range* particionisanje tabele koja sadrži sve preglede i operacije. *Range* particionisanje podrazumeva podelu podataka u tabeli na osnovu zadatog opsega vrednosti, uglavnom datuma. U našem slučaju, napravili bismo particiju sa pregledima i operacijama zakazanim za narednih 6 meseci, oslanjajući se na pretopostavku da se pregledima i operacijama iz tog perioda najčešće pristupa. Na taj način se u proverama da li je slobodan doktor, sestra i sala razmatraju samo pregledi iz particije, ako se novi pregled/operacija zakazuje u pomenutih 6 meseci, što u velikoj meri ubrzava izvršenje upita. Na svakih mesec dana, automatski bi se brisala postojeća i kreirala nova particija za narednih 6 meseci pomoću *PostgreSQL trigger*-a koji bi pozivao funkciju za to kada je uslov navršenja mesec dana zadovoljen. U tom kratkom periodu, svi upiti bi bili izvršavani nad celom tabelom. Takođe zbog velikog broja zakazanih pregleda i operacija, primenili bismo *list* particionisanje da napravimo particiju koja bi sadržala samo preglede i operacije koje administratori trebaju da odobre do kraja dana. Razlog je velika potreba za dobavljanjem tih pregleda i operacija, a koji čine mali procenat svih postojećih pregleda i operacija u bazi podataka, jer su to samo zahtevi kreirani u tekućem danu. *List* particionisanje podrazumeva podelu na osnovu liste poznatih vrednosti, kao što je enumeracija u našem slučaju.

Da bismo dodatno povećali skalabilnost naše aplikacije, možemo uvesti keširanje učitanih podataka u našoj aplikaciji. *Spring Hibernate* podrazumevano vrši L1 keširanje, koje nam omogućava da, unutar jedne sesije, svaki zahtev za objektom iz baze uvek vraća istu instancu objekta. Takođe, postoji i L2 keširanje, koje se odnosi na privremeno čuvanje objekata dobijenih iz baze na serveru, tako da se pri novom upitu ka bazi prvo proverava da li se traženi objekti već nalaze u *cache*-u. U našoj aplikaciji bismo, na primer, mogli keširati informacije o klinikama, zbog velikog broj pacijenata

koji zakazuju preglede i pri tome pregledaju detalje tih klinika. Takođe, detalji o klinici se neće previše često menjati, pa su i sa tog aspekta pogodni za keširanje. Konfigurisanjem TTL (*time to live*) se može odrediti dužina trajanja ovog *cache*-a i time balansirati između opterećenja baze i ažurnosti podataka.

U cilju dodatnog poboljšanja sistema, mogli bismo nadgledati kako doktori i sestre postupaju prilikom neuspešnog kreiranja zahteva za godišnji odmor/odsustvo, zbog preklapanja sa nekim pregledom ili operacijom zakazanim u tom periodu. Ako se uoči da u tom slučaju većina pomenutih korisnika ode da pogleda šta imaju zakazano u tom periodu kako bi odredili datum za koji su slobodni i mogu da kreiraju zahtev, onda bi bilo dobro olakšati im taj postupak. Na primer, mogli bismo im izlistati sve preglede/operacije u periodu koji obuhvata specificirani period odmora/odsustva odmah prilikom neuspešnog kreiranja zahteva. Takođe, mogli bismo posmatrati i koji procenat korisnika upotrebljava pretragu, tamo gde je ona dostupna kao alternativni način traženja entiteta, i koliko često, pa na osnovu toga zaključiti da li je potrebno dodati pretragu i za stranice kod kojih ona nije implementirana. Recimo, ako primetimo da veliki broj administratora klinike koristi pretragu na stranicama na kojima je ona dostupna, onda bi bilo dobro pretragu dodati i na stranice na kojima ona trenutno nije implementirana, a mogla bi biti. To bi mogle biti stranice koje prikazuju kreirane predefinisane preglede, zahteve za godišnjim odmorom/odustvom, dodate medicinske sestre itd.

Okvirna procena neophodne memorije za skladištenje svih podataka u narednih 5 godina je data u tabeli, a izvedena je na osnovu sledećih pretpostavki:

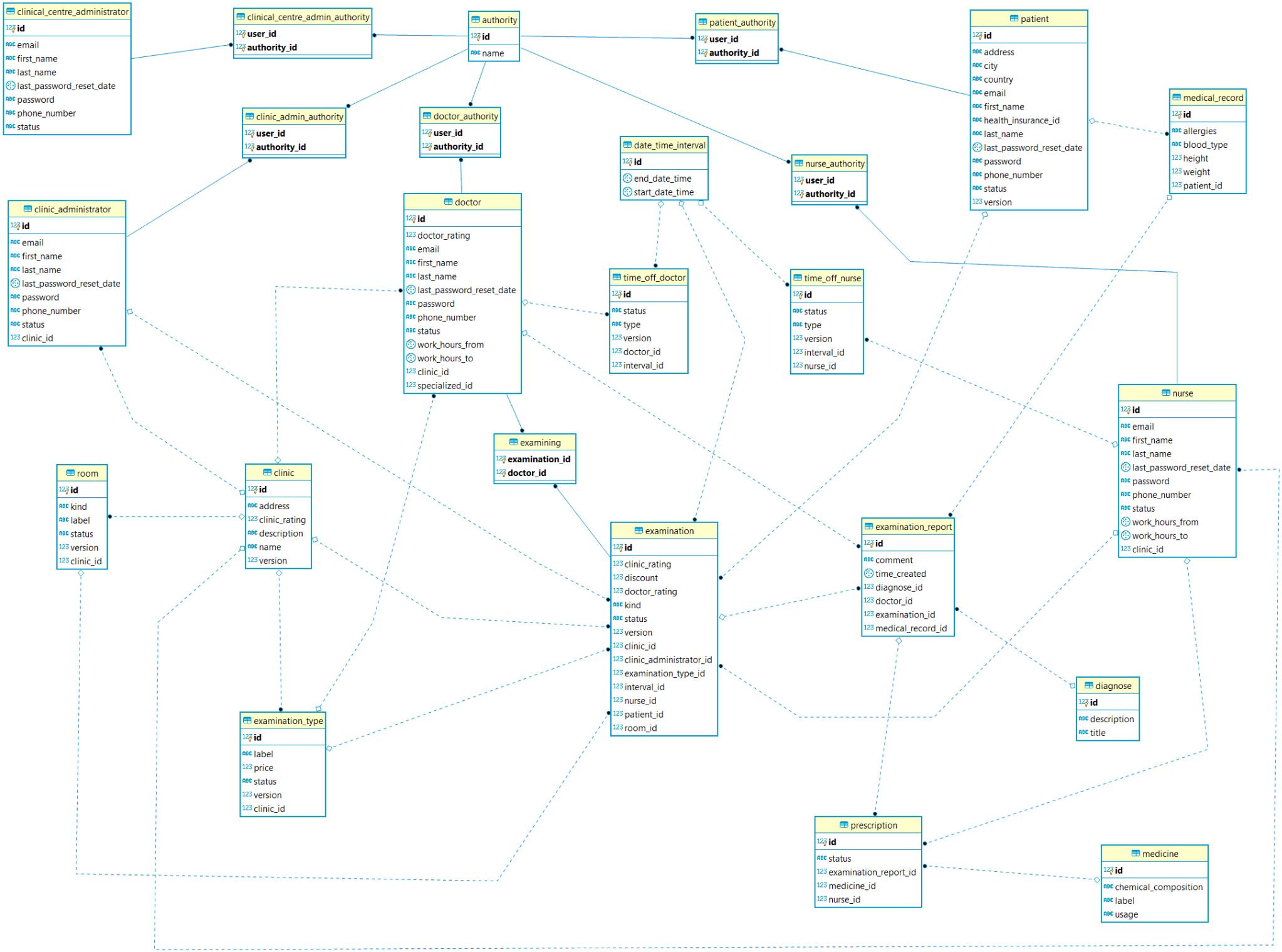
- ukupan broj korisnika je 200.000.000
- na svakih 20.000 pacijenata postoji 1 klinika, pa će postojati 10.000 klinika
- na svakih 50 klinika postoji 1 administrator kliničkog centra, pa će postojati 200 administratora kliničkog centra
- svaka klinika ima 10 sala i u njoj radi 10 doktora, 10 med. sestara i 5 administratora klinike
- kada oduzmemo radnike svih klinika, ostaje 199.750.000 pacijenata u sistemu
- svaki doktor u klinici vrši drugačiji tip pregleda, pa je broj tipova pregleda po klinici jednak broju njenih doktora
- svaki doktor i med. sestra godišnje zatraže 5 puta odmor/odsustvo
- broj zahteva za pregledima i operacijama je mesečno 1.000.000, pa je nakon 5 godina to 60.000.000 pregleda i operacija
- ukupan broj dijagnoza je 500.000
- ukupan broj lekova je 1.000.000
- ukupan broj prepisanih recepata je 100.000.000

Tabela	Veličina prazne tabele (kB)	Veličina jedne torke (kB)	Procena broja torki	Procena zauzeća memorije (kB)
authority	40	0,055	5	40
clinic	16	0,104	10.000	1.056
clinic_administrator	32	0,172	50.000	8.632
clinic_admin_authority	8	0,036	50.000	1.808
doctor	32	0,195	100.000	19.500
doctor_authority	8	0,036	100.000	3.608
nurse	32	0,180	100.000	3.632
nurse_authority	8	0,036	100.000	3.608
patient	40	0,190	199.750.000	37.952.540
patient_authority	8	0,036	199.750.000	7.191.008
medical_record	16	0,067	199.750.000	13.383.266
clinical_centre_administrator	40	0,154	200	70
clinical_centre_admin_authority	8	0,036	200	7
room	24	0,075	100.000	7.524
examination_type	16	0,083	100.000	8.316
date_time_interval	8	0,044	100.000.000	4.400.008
examination	16	0,122	60.000.000	7.320.016
examining	8	0,036	60.000.000	2.160.008
diagnose	24	0,500	500.000	250.024
medicine	24	0,500	1.000.000	500.024
time_off_doctor	16	0,067	500.000	33.516
time_off_nurse	16	0,067	500.000	33.516
examination_report	16	0,083	60.000.000	4.980.016
prescription	8	0,067	100.000.000	6.700.008
			Ukupno u kB	**84.961.751**
			Ukupno u GB	**~81**

Pošto je procena da će za period od 5 godina nastati oko 81 GB podataka, dovoljan je jedan server baze podataka koji će te podatke skladištiti.

Dizajn šeme baze podataka

Na sledećoj strani nalazi se prikaz šema baze podataka aplikacije.


clinical_centre_administrator
id
email
first_name
last_name
last_password_reset_date
password
phone_number
status
clinical_centre_admin_authority
user_id
authority_id
authority
id
name
patient_authority
user_id
authority_id
patient
id
address
city
country
email
first_name
health_insurance_id
last_name
last_password_reset_date
password
phone_number
status
version
medical_record
id
allergies
blood_type
height
weight
patient_id
clinic_admin_authority
user_id
authority_id
doctor_authority
user_id
authority_id
date_time_interval
id
end_date_time
start_date_time
nurse_authority
user_id
authority_id
clinic_administrator
id
email
first_name
last_name
last_password_reset_date
password
phone_number
status
clinic_id
doctor
id
doctor_rating
email
first_name
last_name
last_password_reset_date
password
phone_number
status
work_hours_from
work_hours_to
clinic_id
specialized_id
time_off_doctor
id
status
type
version
doctor_id
interval_id
time_off_nurse
id
status
type
version
interval_id
nurse_id
nurse
id
email
first_name
last_name
last_password_reset_date
password
phone_number
status
work_hours_from
work_hours_to
clinic_id
examining
examination_id
doctor_id
room
id
kind
label
status
version
clinic_id
clinic
id
address
clinic_rating
description
name
version
examination
id
clinic_rating
discount
doctor_rating
kind
status
version
clinic_id
clinic_administrator_id
examination_type_id
interval_id
nurse_id
patient_id
room_id
examination_report
id
comment
time_created
diagnose_id
doctor_id
examination_id
medical_record_id
diagnose
id
description
title
examination_type
id
label
price
status
version
clinic_id
prescription
id
status
examination_report_id
medicine_id
nurse_id
medicine
id
chemical_composition
label
usage

Kompletan crtež dizajna predložene arhitekture