Testiranje performansi

Katarina Vučić SV29/2020

Testiranja su obavljena na računaru sa procesorom Intel i7 10. generacije i 8 jezgara. Računar takođe posjeduje 16 GB RAM memorije. Korišten je JMeter alat za testiranje performansi i brzine softvera.

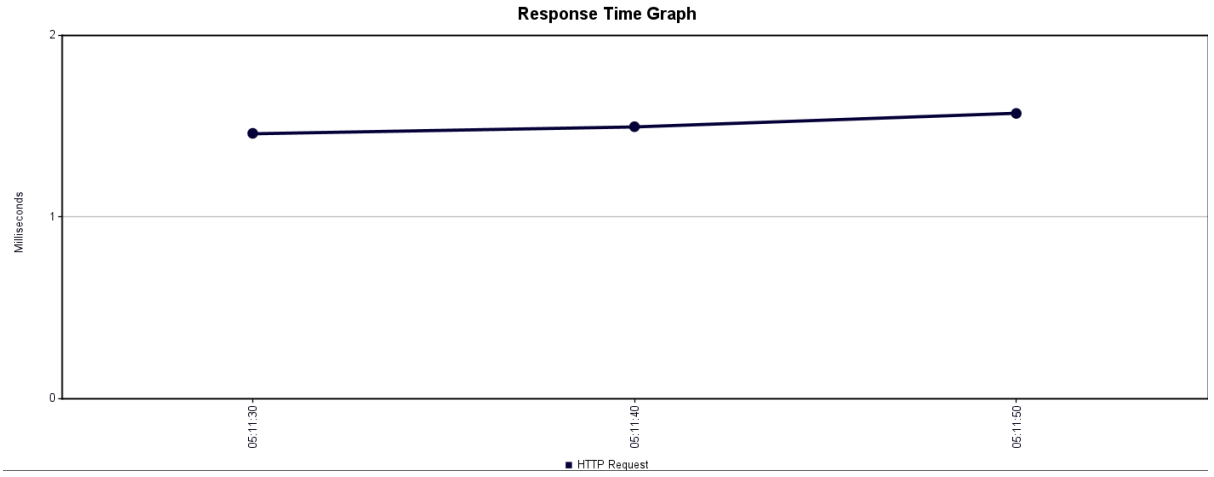
Struktura projekta se sastoji iz React frontend-a, Go Gin backend aplikacije i Go simulacije. Za skladištenje podataka smo koristili MySQL bazu podataka, a serijske (vremenski određene) podatke smo čuvali u InfluxDB.

- **Testiranje opterećenja sistema (*load testing*) za česte scenarije korišćenja**

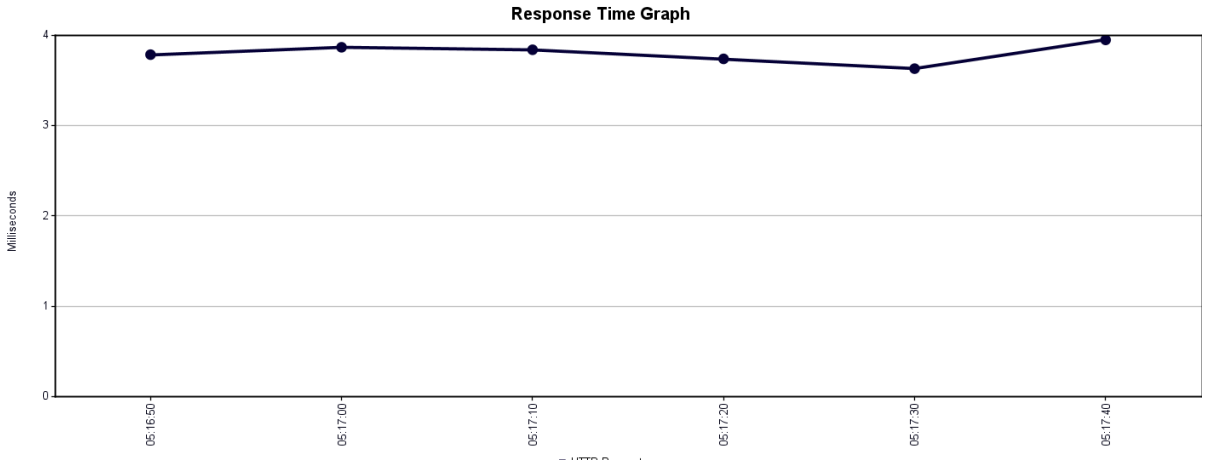
1. GET VEHICLE GATE

Testirali smo vrijeme odziva za dobavljanje Kapije iz baze podataka. Na narednim graficima možemo vidjeti rezltate testova.

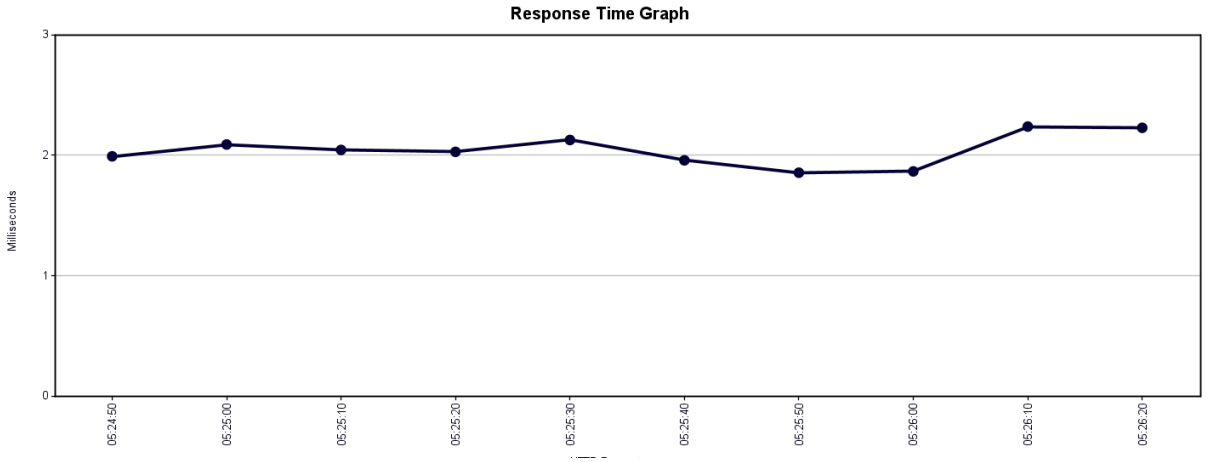
Naredni grafik prikazuje rezltate testova simulirane sa 100 thread-ova, odnosno 100 korisnika. Možemo da primijetimo da su odgovori od backend-a dobijeni za manje od 2 ms, što se smatra odličnim rezultatom.



Testiranje sa 1000 korisnika se takođe prikazalo kao vrlo brzo, ali nešto sporije od prethodnog testiranja. Svi zahtjeva su završeni za manje od 4ms. Rezultati su ponovo bili odlični. Zaključak je da je pisanje čistog SQL koda, bez biblioteka koje automatski generišu upite ka bazi, znatno ubrzalo proces dobavljanja podataka iz baze.



Da bi dodatno opteretili sistem, testirali smo i sa 2000 korisnika, koji su u periodu od 10 sekundi, slali po 100 zahtjeva (po korisniku). Rezultate možemo da vidimo na grafiku ispod. Možemo da vidimo da su zahtjevi poprilično brzo izvršavani, međutim, ukupno je su trebale više od 2 minute da se ovaj test završi.



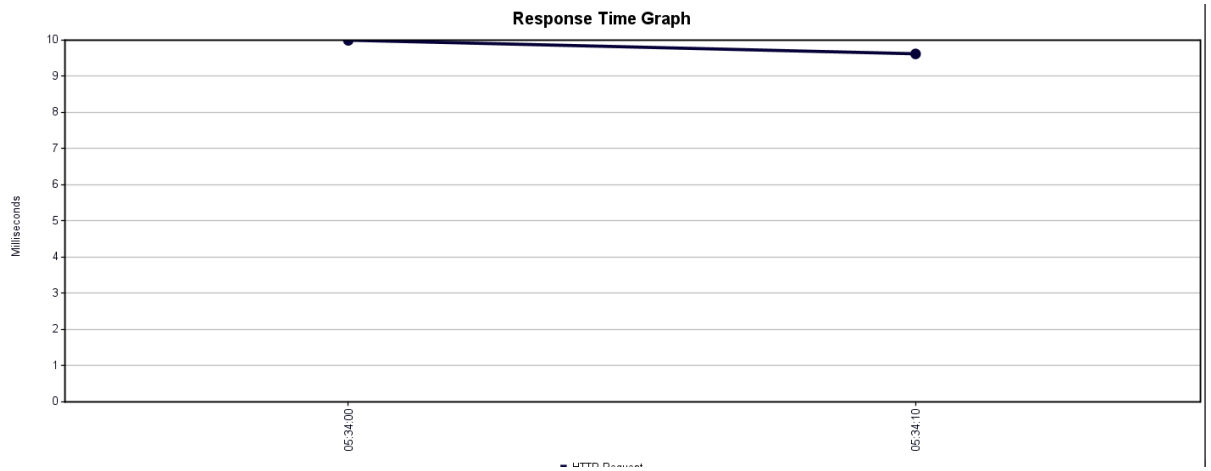
Zaključak je da je pisanje čisto SQL koda dovelo do odličnih performansi dobavljanja podataka iz baze i da se Go backend pokazao kao odličan za potrebe našeg sistema.

2. GET SPRINKLER DATA

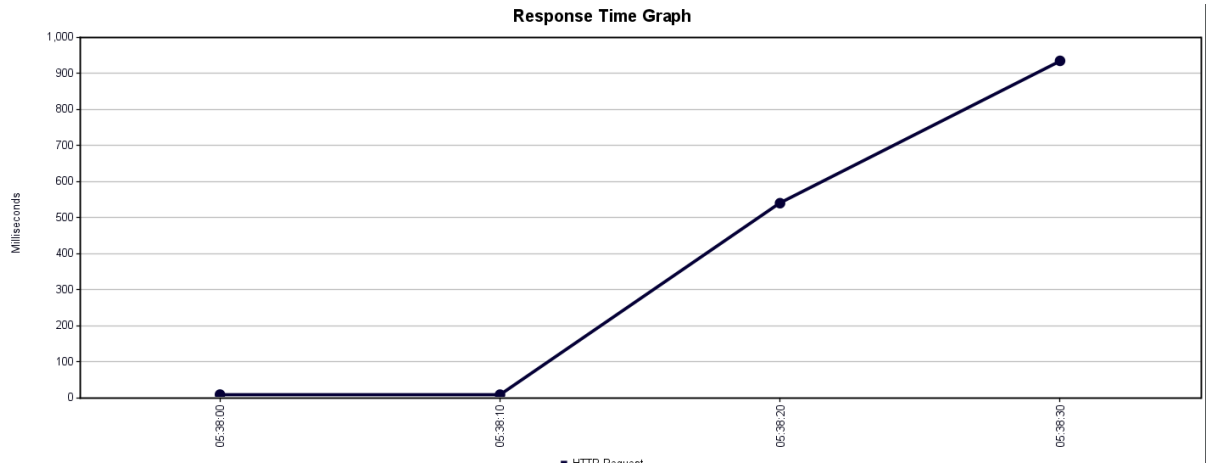
U narednom tekstu ćemo opisivati testove obavljenje nad InfluxDB-om, konkretno operacija dobavljanja iz baze.

Na samom početku smo rad testirali sa 100 korisnika, koji su u roku od 10 sekundi slali po 20 zahtjeva. Kao što smo i očekivali, dobili smo nešto lošije, ali ipak zadovoljavajuće rezulate. Rezultati su lošiji jer Influx baza radi sa daleko većom količinom podataka od MySQL baze, ali su zadovoljavajući jer smo sva dobavljanja iz Influx baze rješavali pomoću Flux query-ja, koji su se pokazali kao najbolje rješenje. Ručno filtriranje podataka, kroz Go kod, je loša

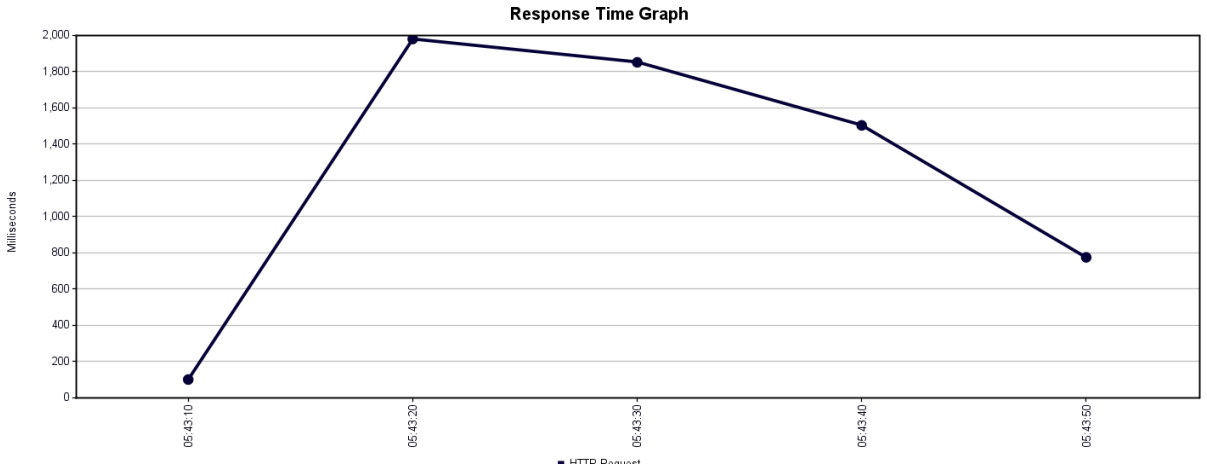
praksa, pa se tom metodom nismo bavili, ukoliko baš nismo morali. Na grafiku ispod vidimo i rezltate ovog testa. Svi upiti su obavljeni unutar 10ms.



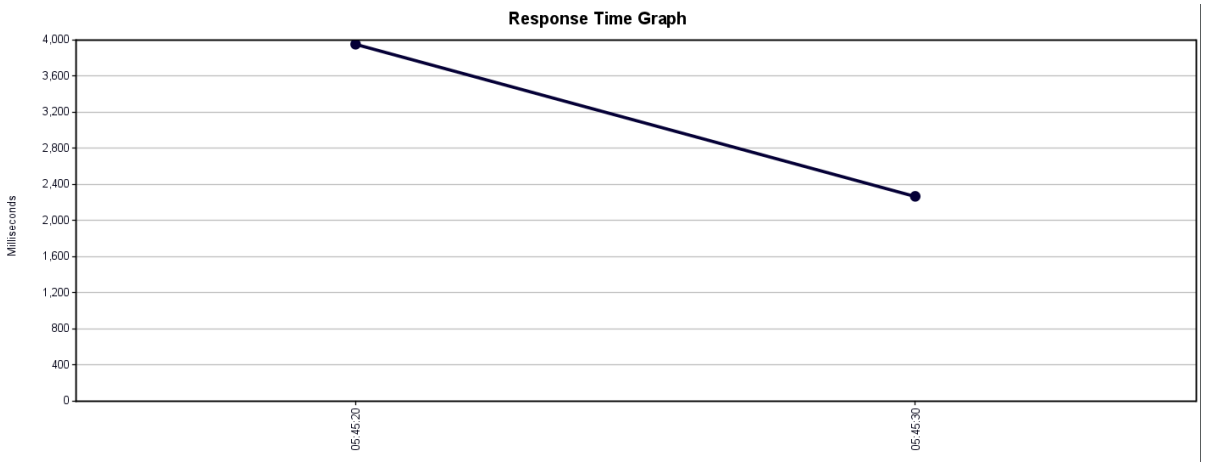
Naredni test smo obavili sa 500 korisnika, koji su unutar 10 sekundi slali po 20 zahtjeva. Na grafiku ispod možemo da vidimo rezultate ovog testa. Svi upiti su obavljeni unutar 1000ms. Prva dva point-a na grafu su odrađena sa 500 korisnika koji su slali samo po jedan zahtjev. Na taj način možemo da uporedimo lijevu i desnu stranu grafa i primijetimo značajnu degradaciju performansi u odnosu na broj zahtjeva po korisniku (na desnoj strani svaki korisnik šalje po 20 zahtjeva). Međutim, i sa ovim opterećenjem, sistem daje odlične performanse.



Naredni test je urađen sa 1000 korisnika, svaki korisnik unutar 10 sekundi pošalje 20 zahtjeva. Rezultati su ponovo zadovoljavajući.

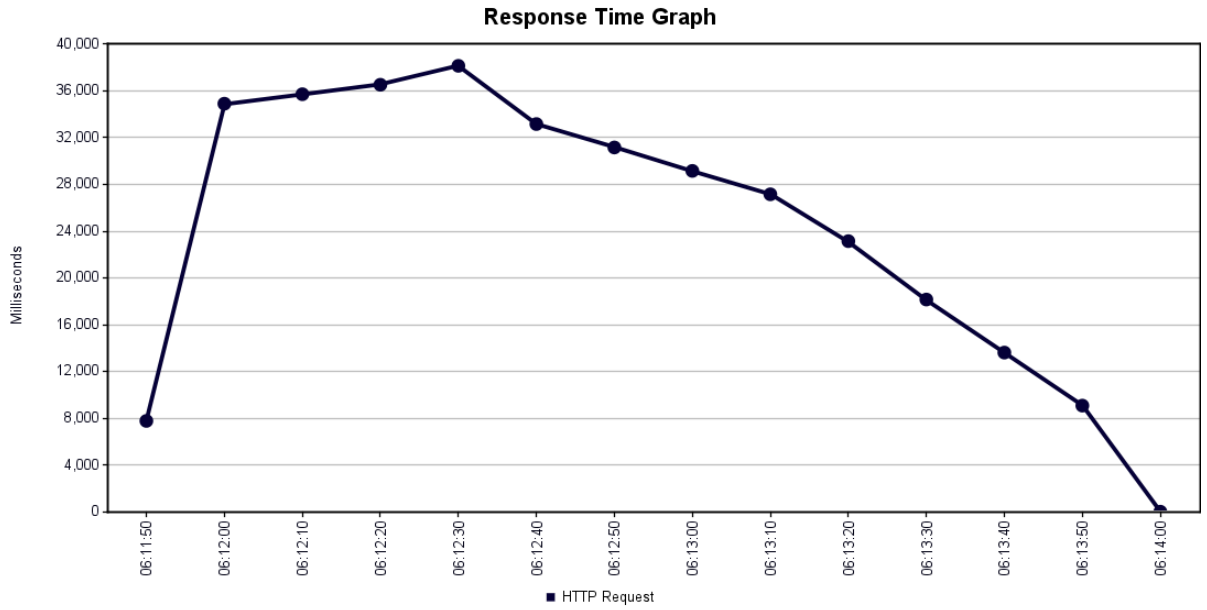


Dodatno smo testirali i sa 2000 korisnika, koji su unutar 10 sekundi slali po 20 zahtjeva. Rezultate testa vidimo na grafiku ispod. Performanse su očekivano duplo lošije, od prethodnog testa.

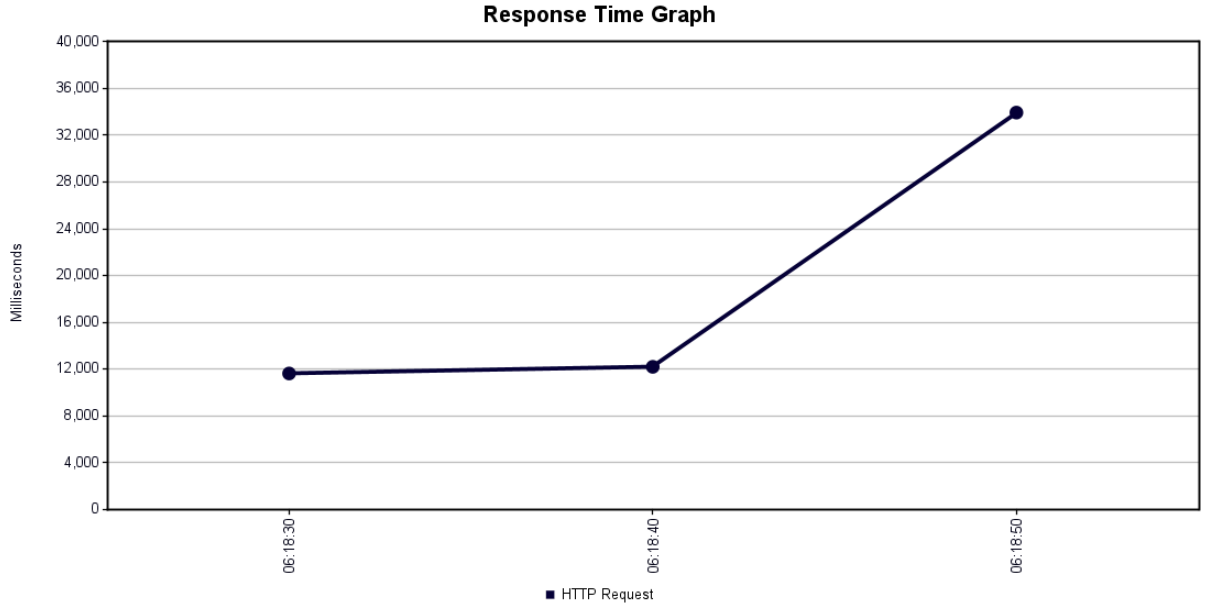


3. PUT SPRINKLER ON

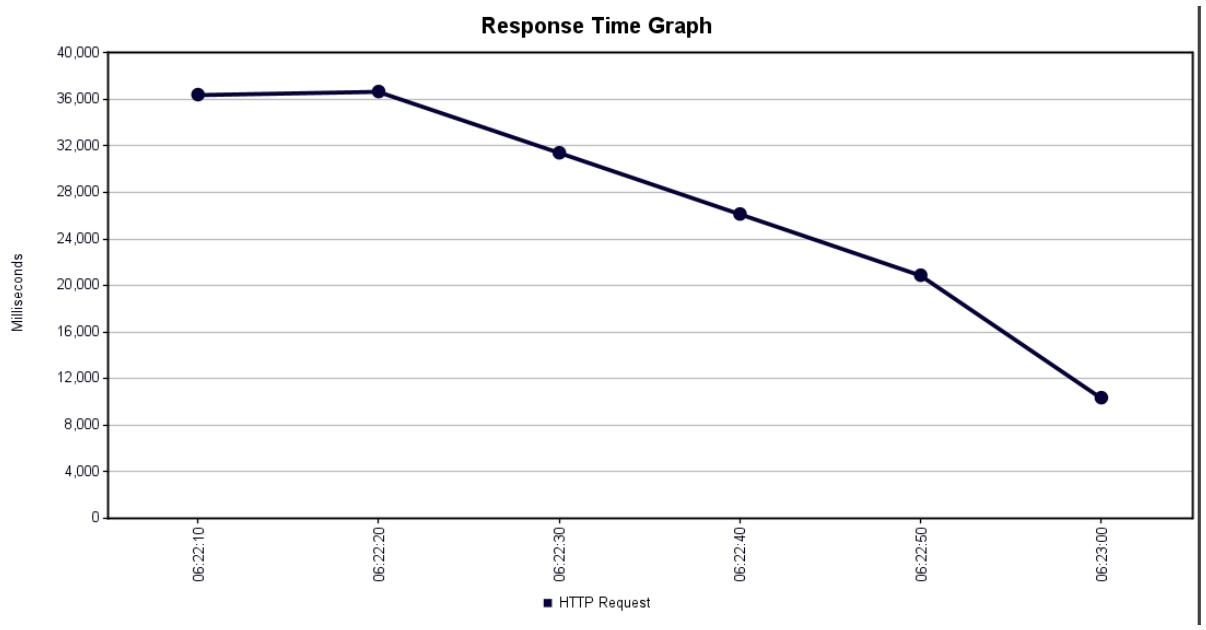
U narednom tekstu testiramo upis podataka u Influx bazu. Prvi test smo obavili sa 100 korisnika, koji su zahtjeve slali u roku od 10 sekundi.



Naredni test smo odradili sa 500 korisnika. Rezultate vidimo na grafiku ispod.

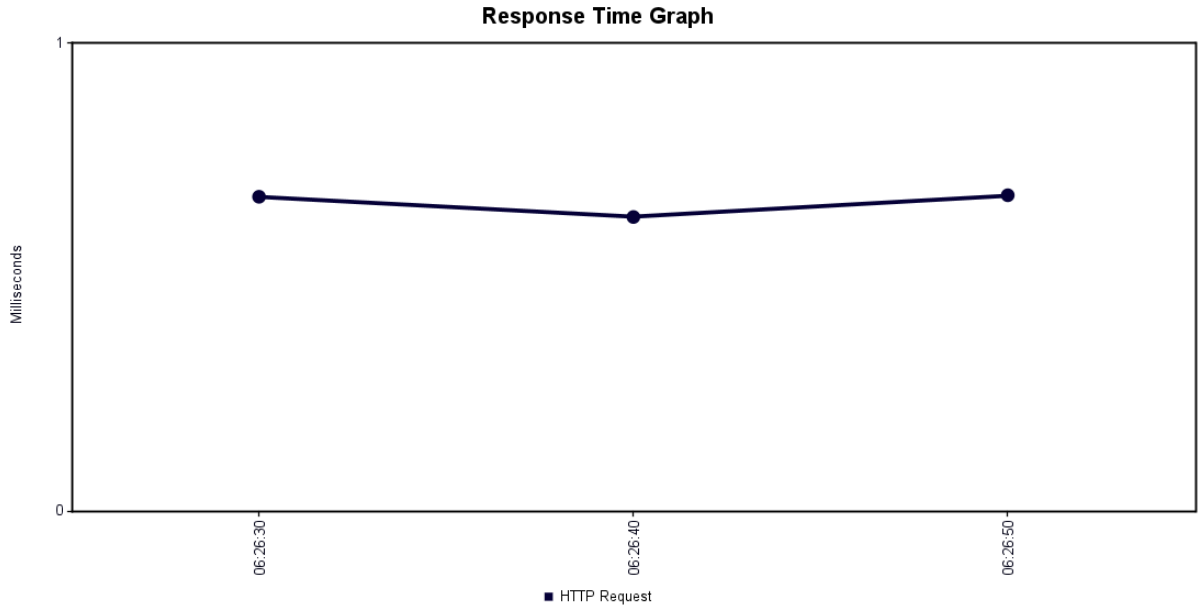


Uradili smo još jedan test, sa 1000 korisnika. Prvi put smo mogli primijetiti degradaciju performansi na backend-u, koji sada dosta usporeno obrađuje zahtjeve. Ovo bi mogli popraviti premiještanjem upisa u Influx bazu u poseban thread, što je vrlo jednostavno implementirati u Go jeziku.

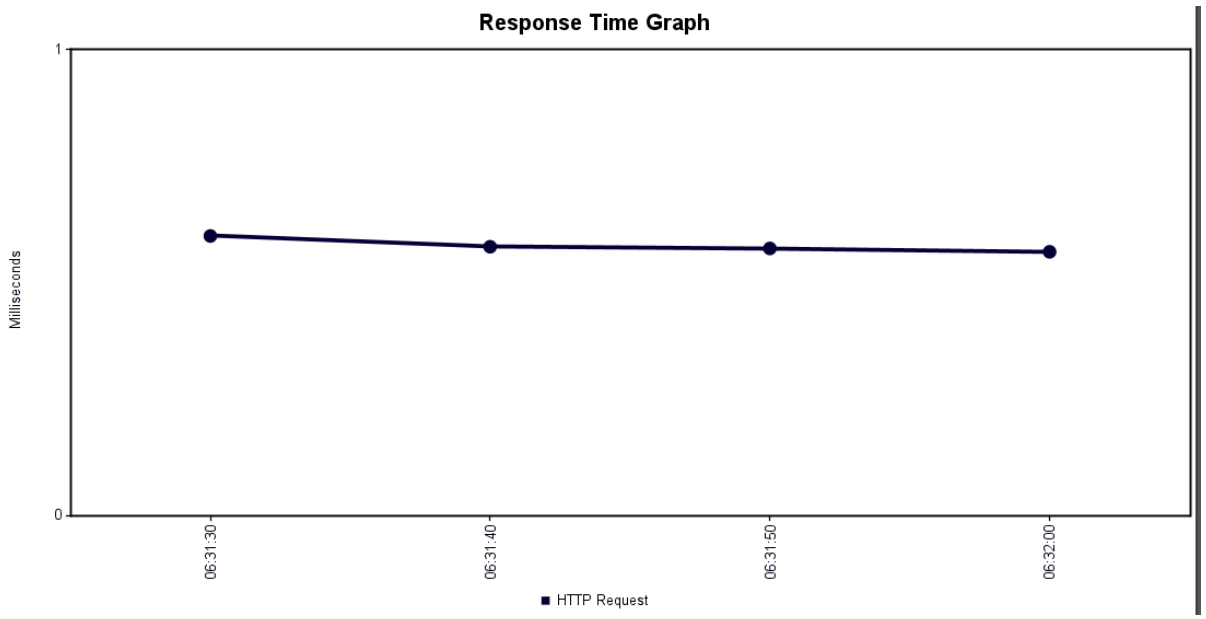


4. GET LAMP DATA

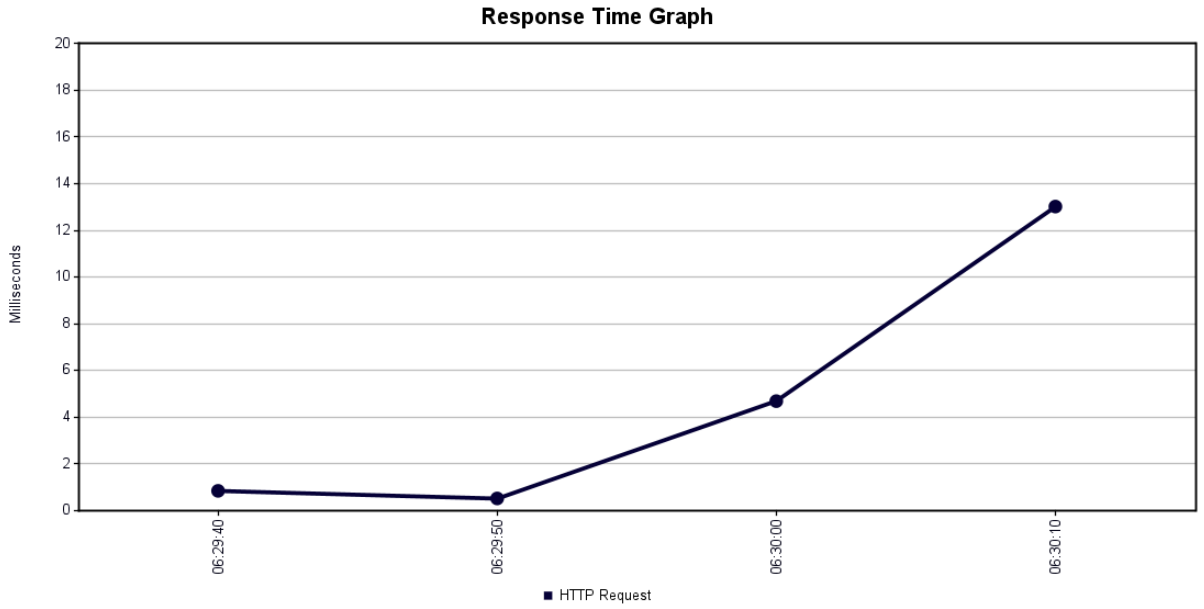
Dobavljamo podatke o lampi iz MySQL baze. Prvi test smo radili sa 100 korisnika, koji su u roku od 10 sekundi slali po 20 zahtjeva. Rezultati testa se nalaze na grafiku i ponovo pokazuju veliku prednost pisanja čistih sql upita.



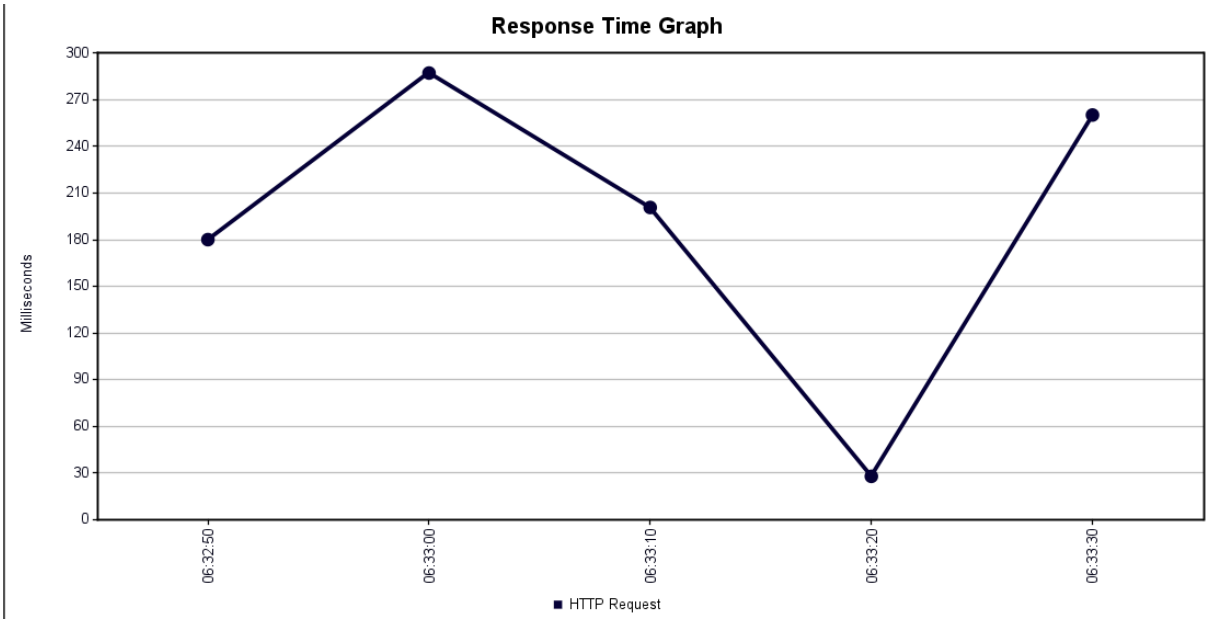
500 korisnika, 10 sekundi za 20 zahtjeva po korisniku.



1000 korisnika, u 10 sekundi 20 zahtjeva po korisniku.



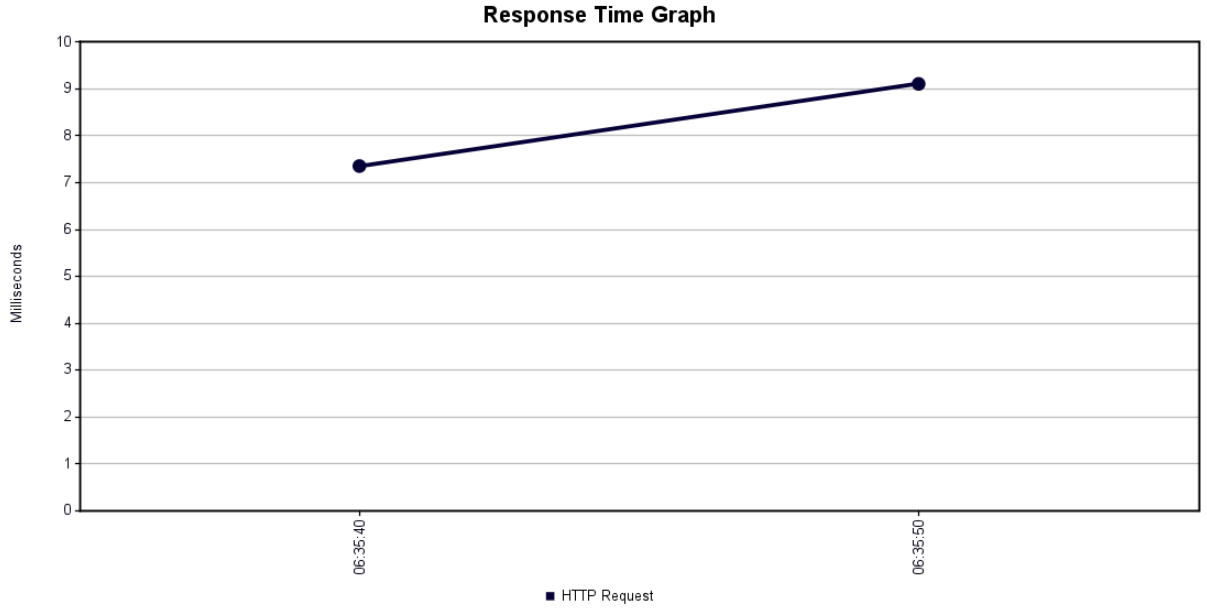
2000 korisnika, koji su za 10 sekundi poslali po 20 zahtjeva. Rezultati su više nego zadovoljavajući.



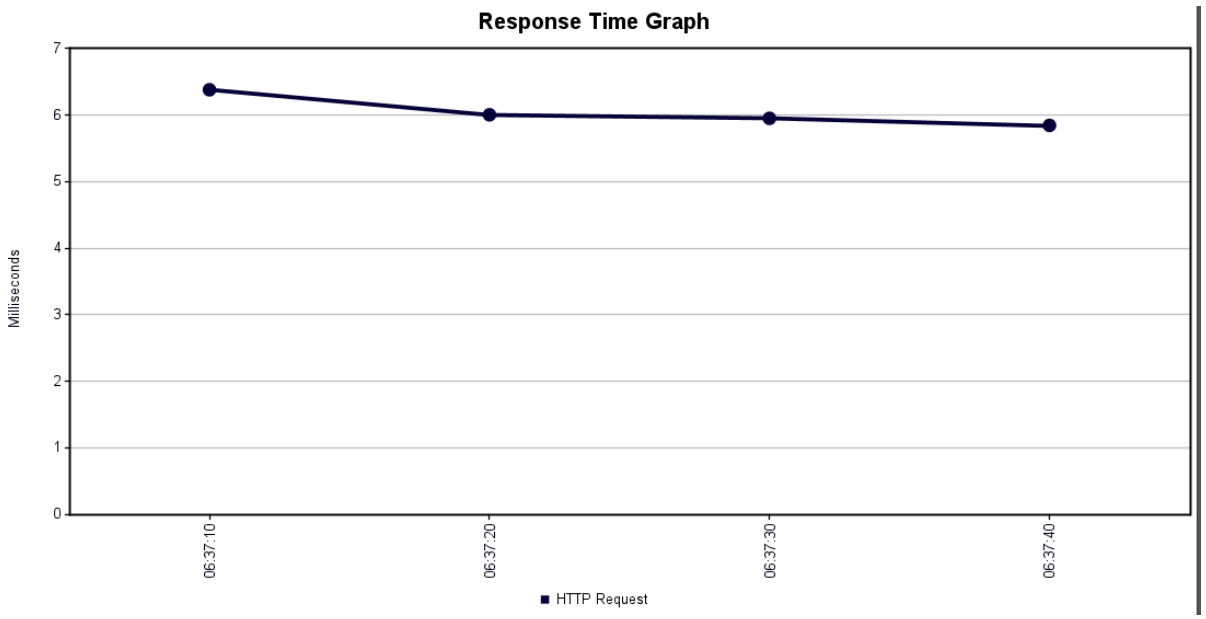
5. POST LAMP

Sada ćemo testirati performanse sistema prilikom dodavanja podataka u MySQL bazu.

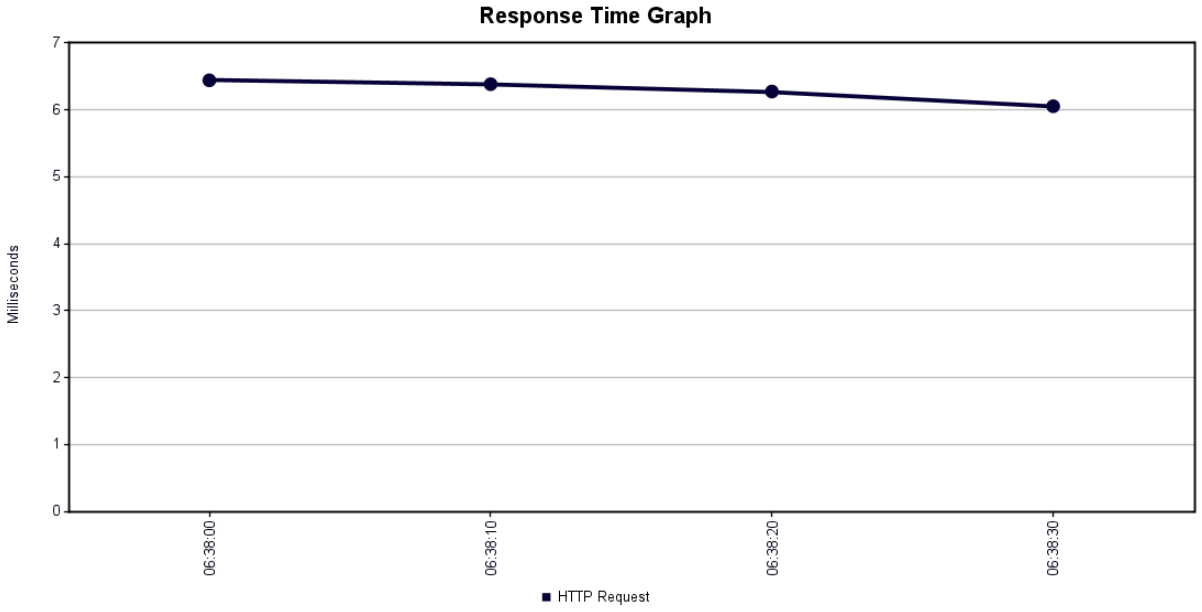
100 korisnika koji šalju zahtjeve u roku od 10 sekundi (10 korisnika po sekundi).



500 korisnika za 10 sekundi.



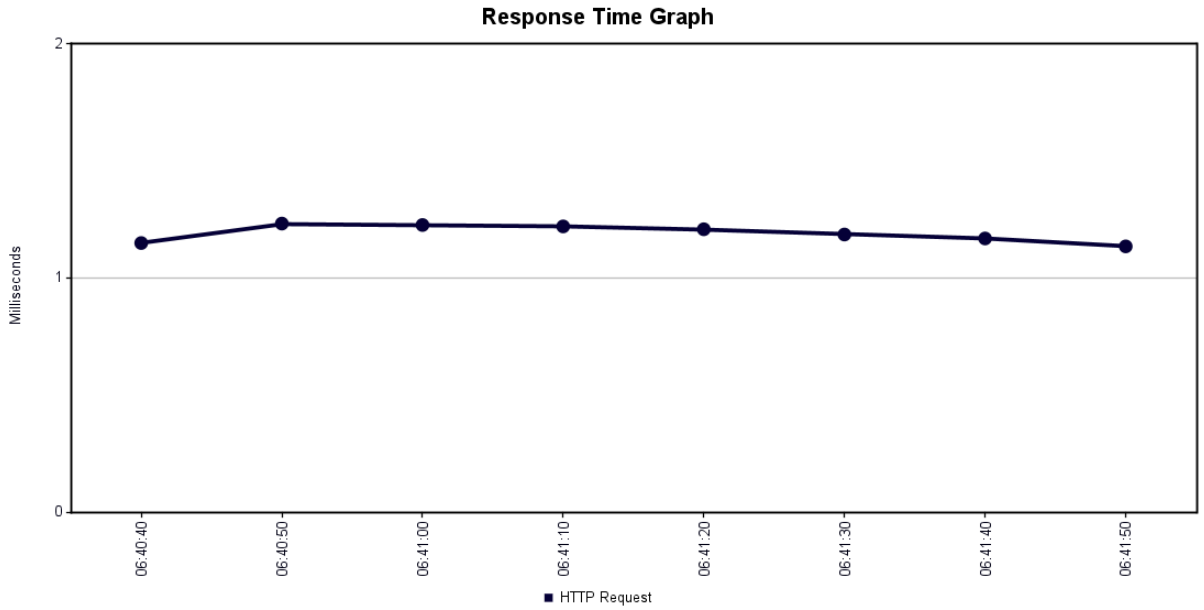
1000 korisnika za 10 sekundi.



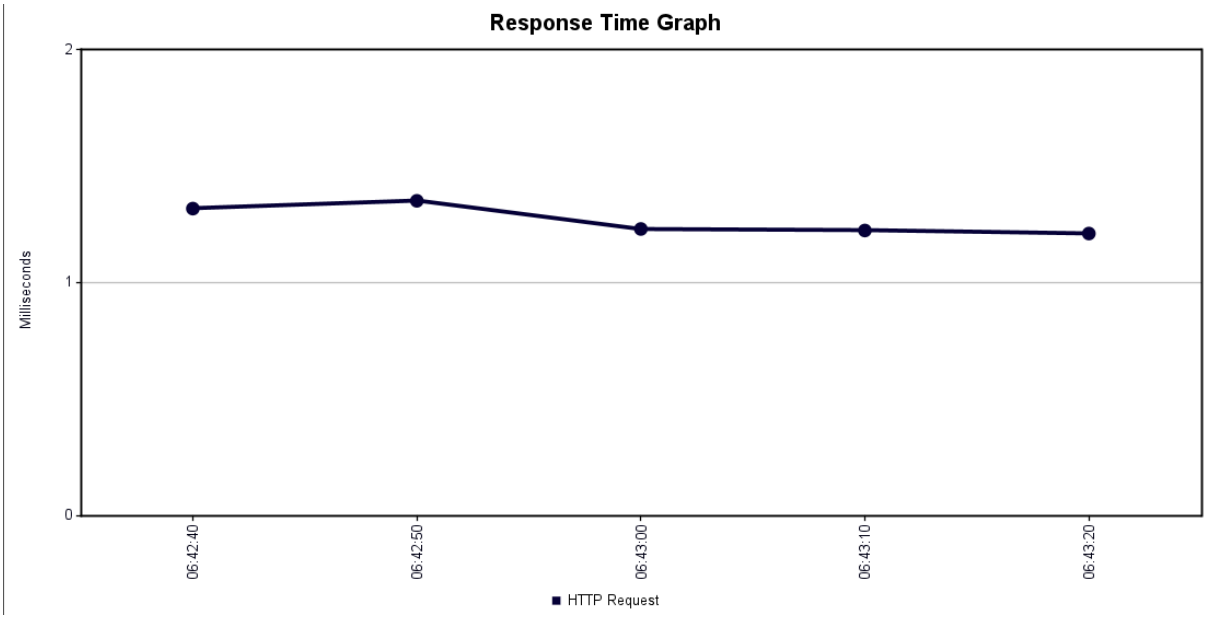
Sada već možemo da zaključimo da sve operacije nad MySQL bazom daju odlične rezultate.

6. PUT VEHICLE GATE TO PUBLIC

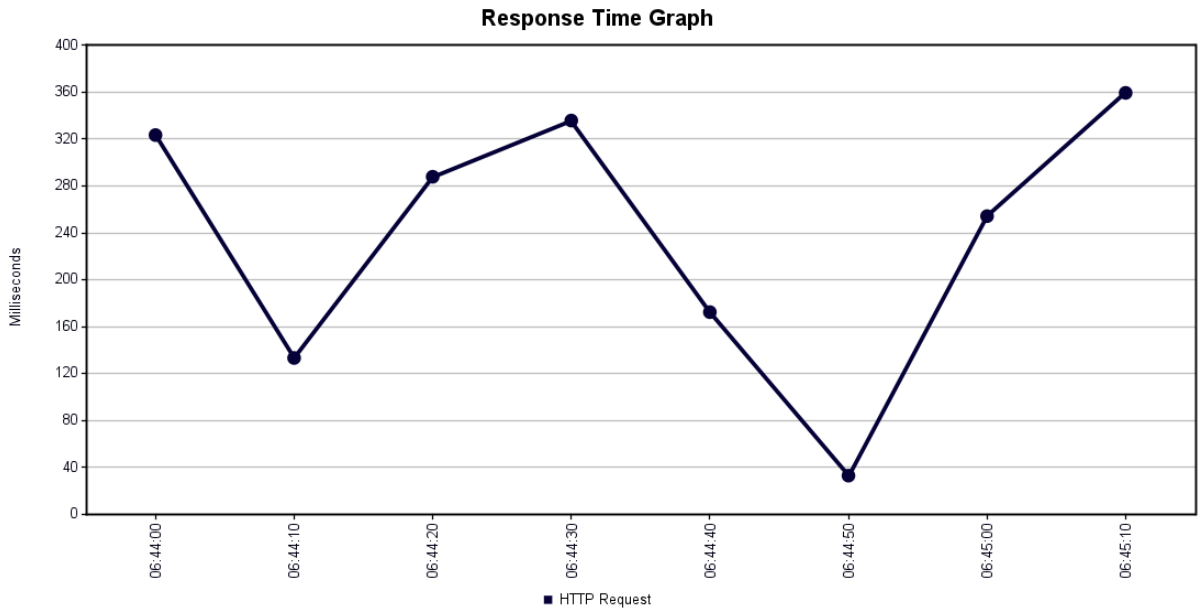
Testiramo brzinu izvršavanja dodavanja u Influx bazu. Prvi test je obavljen sa 100 korisnika koji su u roku od 10 sekundi poslali po 20 zahtjeva. Ssitem je pokazao visoke performanse, čiji dokaz možemo vidjeti na grafiku ispod. Grafik ispod sadrži prikaz za 3 pokretanja ovog testa.



500 korisnika, koji šalju po 20 zahtjeva u roku od 10 sekundi. Ovaj test smo također obavili u 3 iteracije.

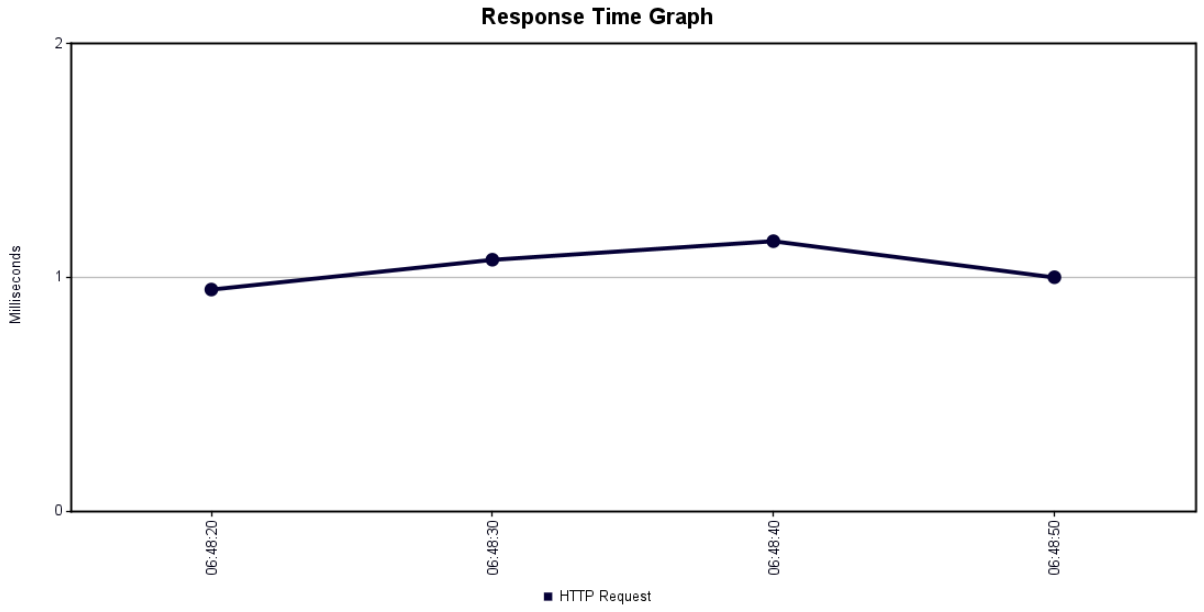


1000 korisnika, koji šalju po 50 zahtjeva u 10 sekundi. I ovaj test je obavljen u 3 iteracije. Kod njega smo mogli primijetiti da se nije mogao uklopiti u 10 sekundi, međutim, njegove performanse su i dalje vrlo zadovoljavajuće.

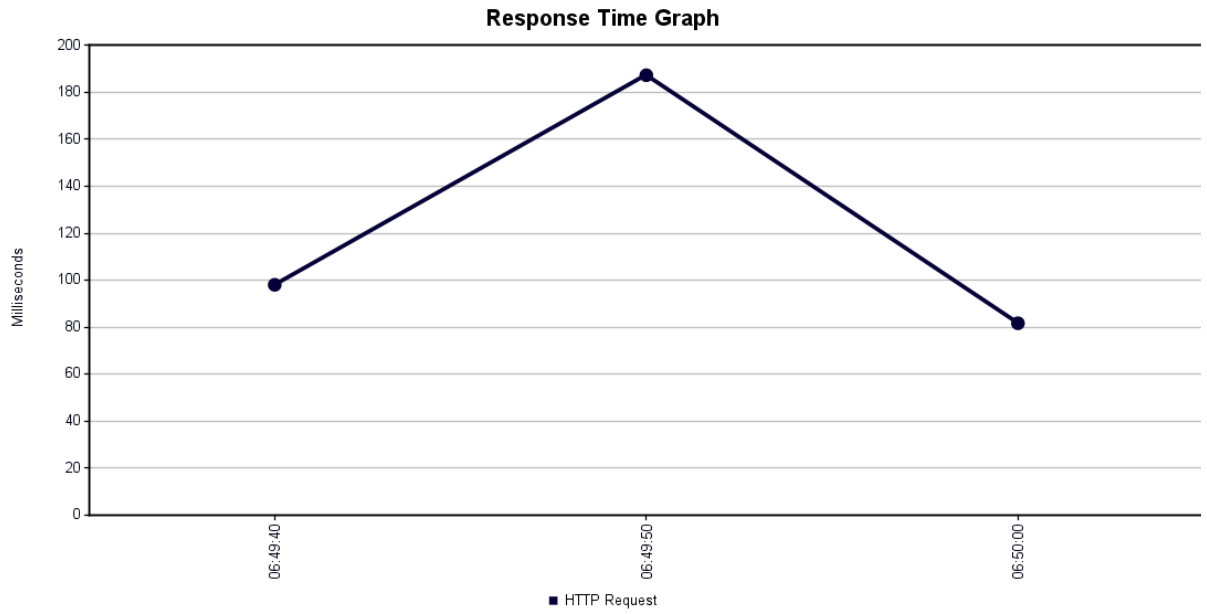


7. GET REAL ESTATE

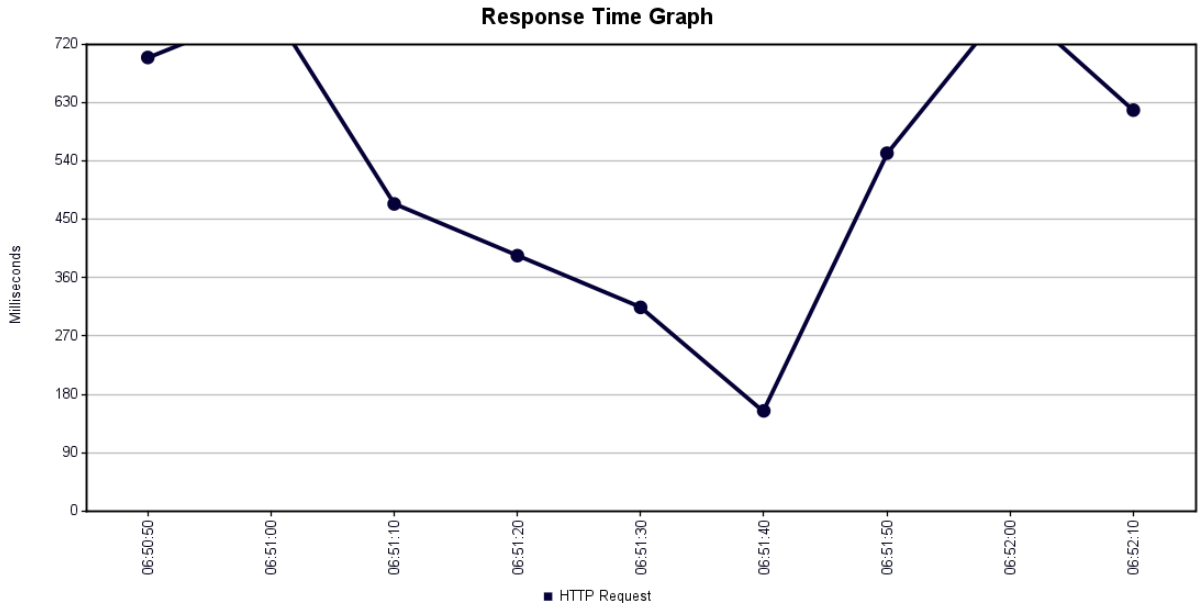
Testiramo dobavljanje nekretnine iz baze podataka. Prvi test je obavljen sa 500 korisnika, koji su u roku od 10 sekundi poslali po 50 zahtjeva. Test je obavljen u 3 iteracije.



1000 korisnika, po 50 zahtjeva, 10 sekundi, 2 iteracije.



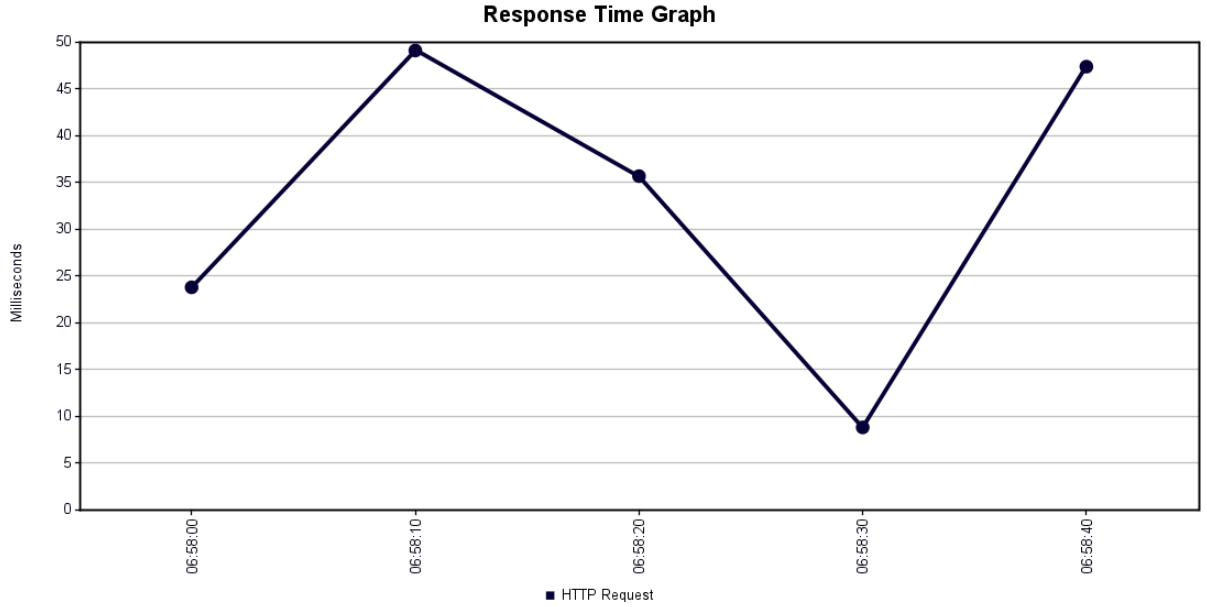
2000 korsinika, po 50 zahtjeva, 10 sekundi, 3 iteracije. Za ovaj test je bilo potrebno mnogo više vremena u odnosu na prethodne testove. Takođe primijtili smo i malo opterećenje mašine na kojoj je test odrađen. Imali smo problem sa vizualizacijom grafa, ali ispravljanje ovog problema, nije bilo u našim rukama, nego JMeter inženjerskog tima. Uprkos tome, možemo da primijetimo odlične performanse sistema i na ovako zahtjevnim testovima.



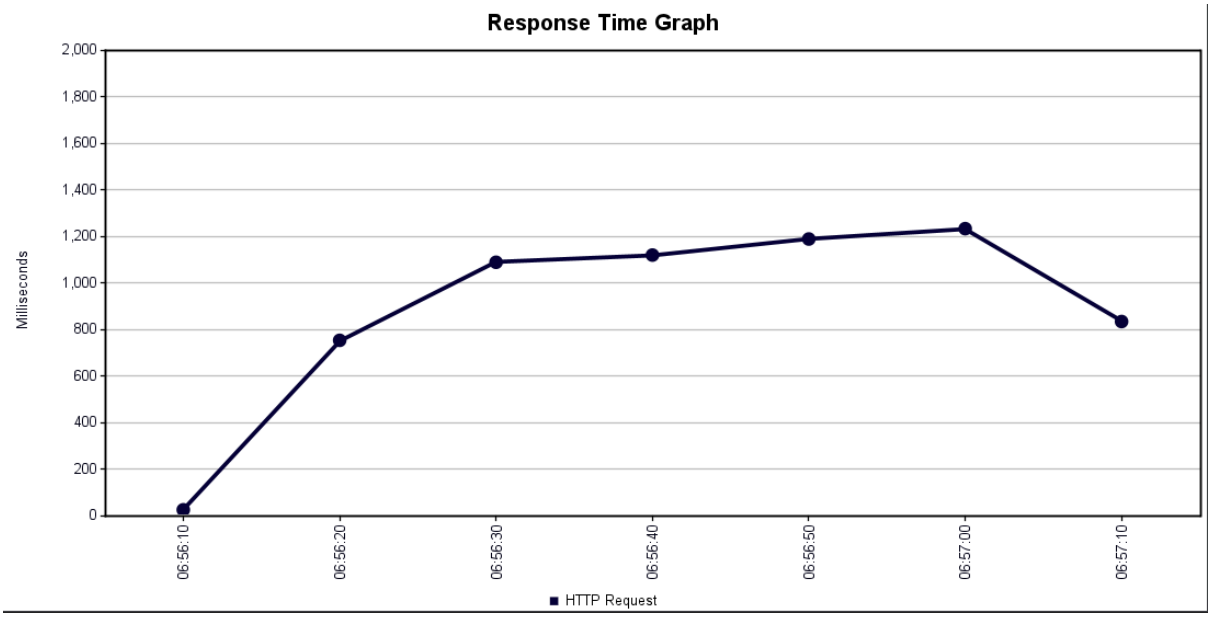
8. GET GRAPH DATA FOR LAMP

Dobavljamo podatke iz Influx baze. Ovi podaci su agregirani kroz Flux upit, zbog čega se očekuju dobre performanse sistema. U prvom testu smo

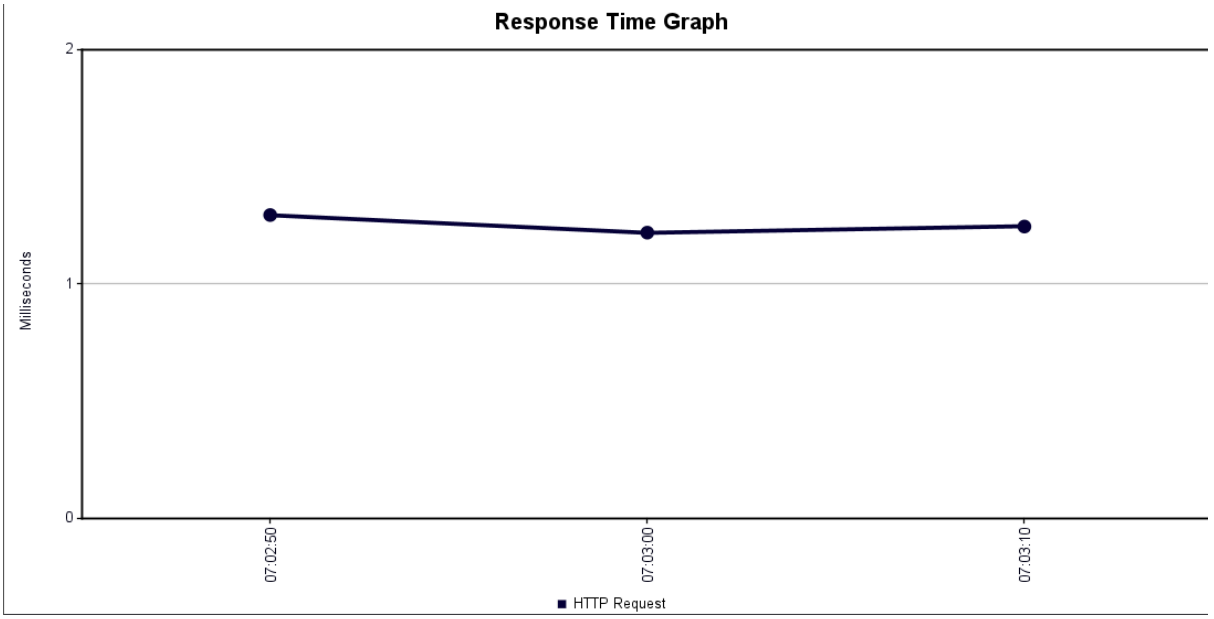
podesili 100 korisnika, koji su za 10 sekundi poslali po 50 zahtjeva. Test je pokrenut u 2 iteracije.



Drugi test je odrađen sa 500 korisnika, koji su u roku od 10 sekundi poslali po 50 zahtjeva.



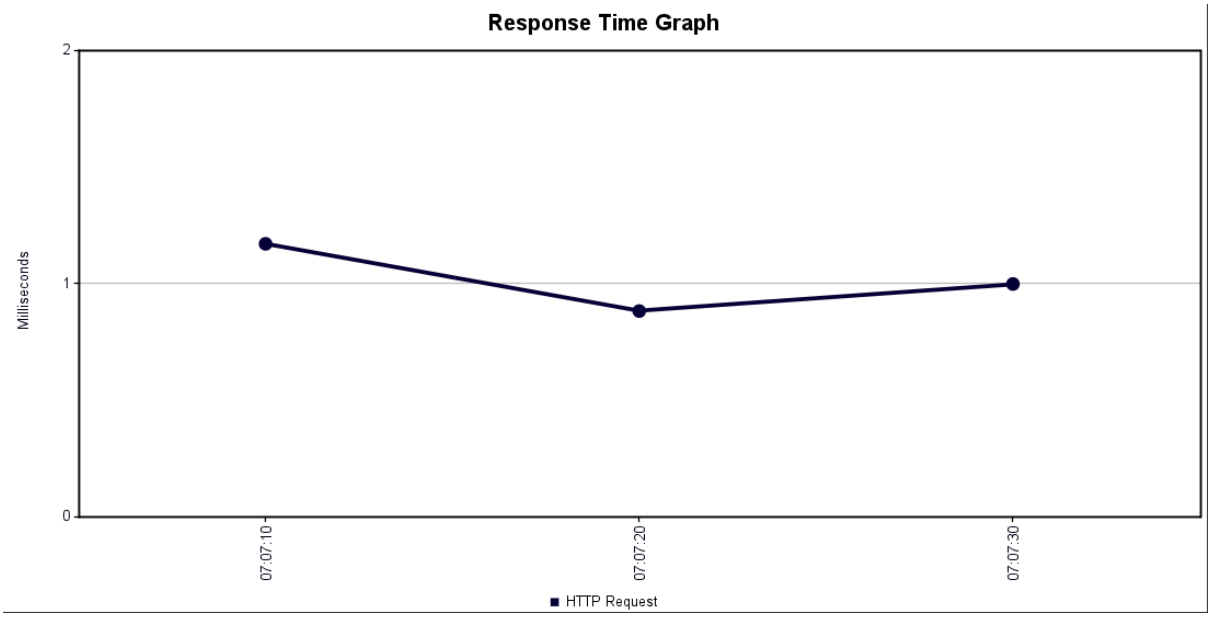
1000 korisnika, 10 sekundi, 2 iteracije pokretanja. Test se uspješno uklopio u 10 sekundi. Ovdje primjećujemo bolje performanse, jer smo po korisniku slali samo 1 zahtjev.



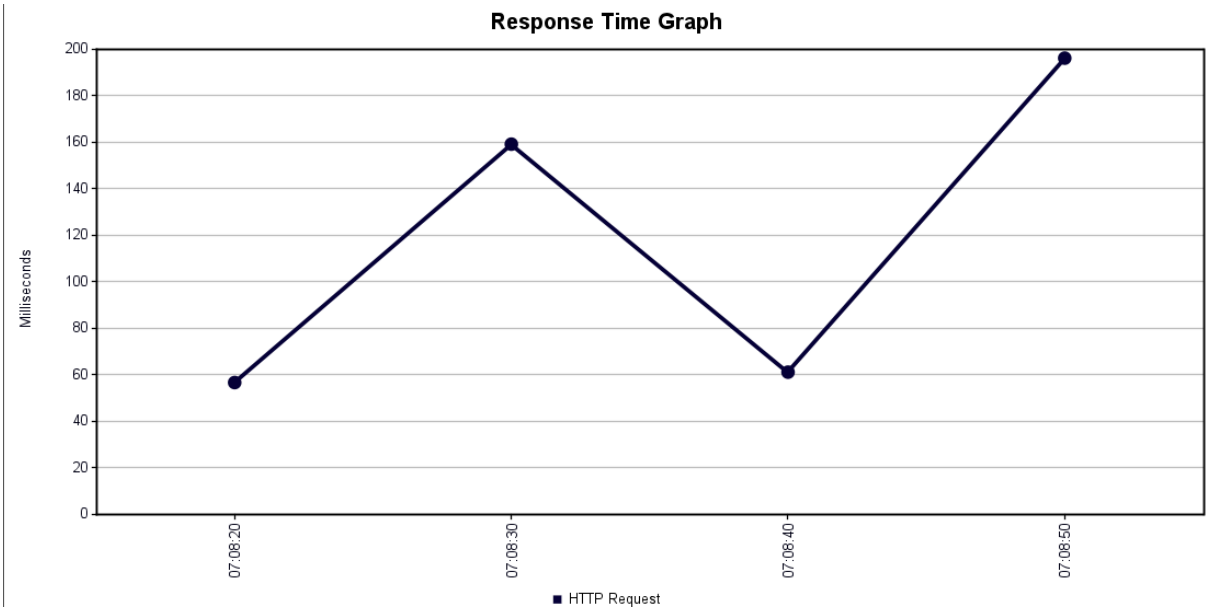
Prilikom testiranja ove funkcionalnosti, primijetili smo otežan rad servera, čiji je uzrok velika količina podataka u Influx bazi.

9. GET PENDING REAL ESTATES

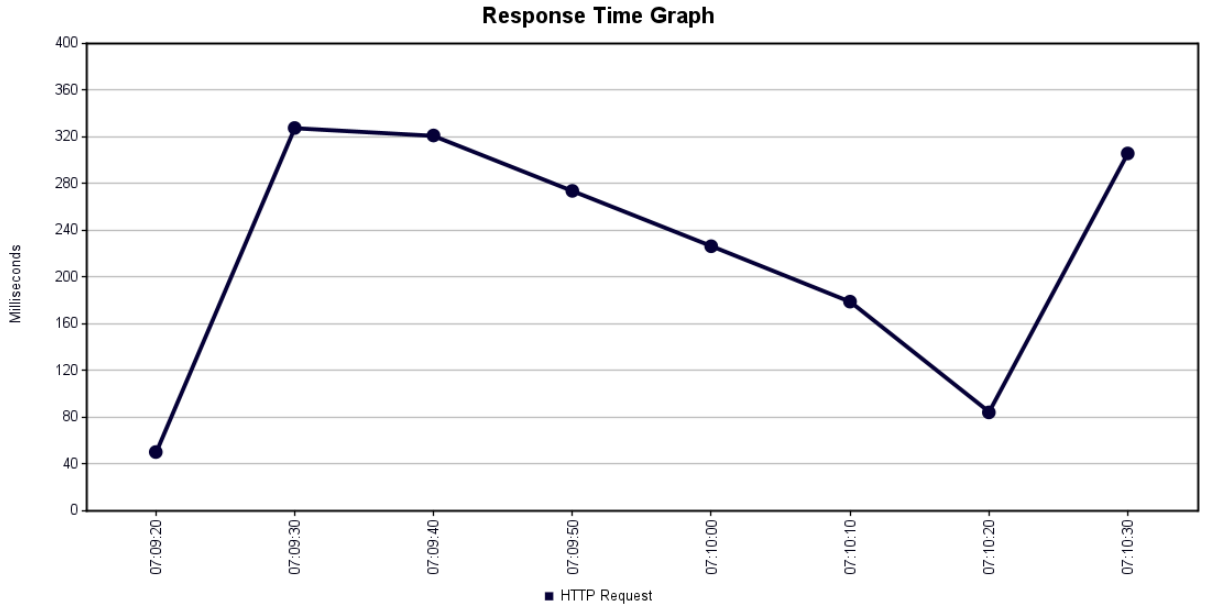
Testiramo scenario u kome admin dobavlja sve nekretnine sa statusom pending. Prvi test je obavljen sa 500 korisnika, 10 sekundi i 50 zahtjeva po korisniku u 3 iteracije.



1000 korisnika, po 50 zahtjeva, 2 iteracije.

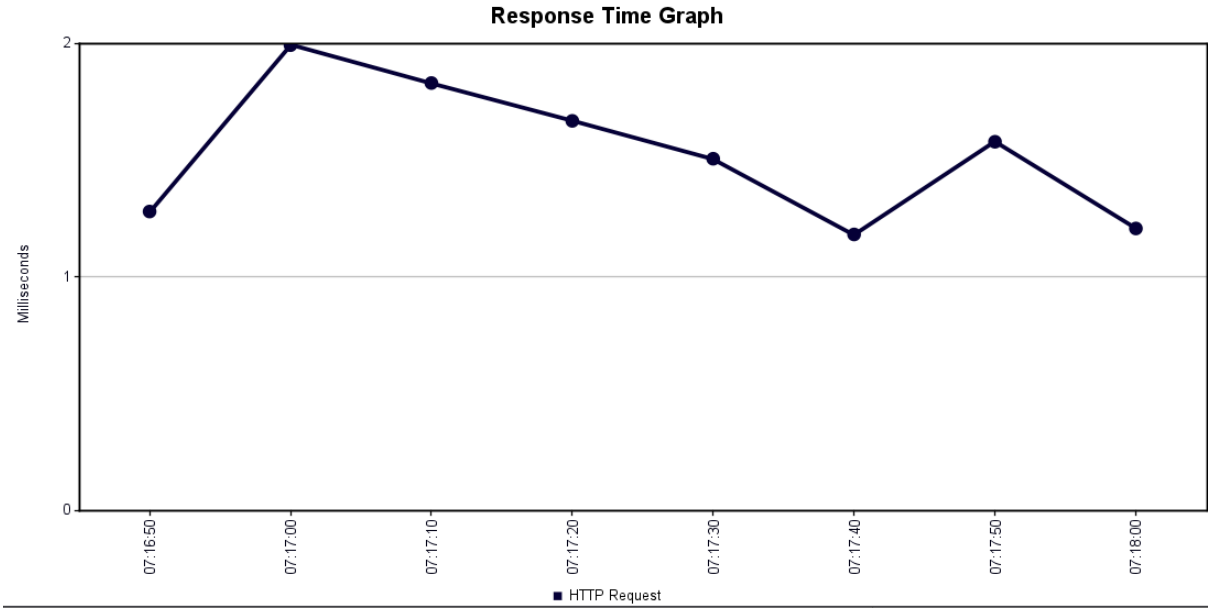


2000 korisnika, po 20 zahtjeva, 2 iteracije.

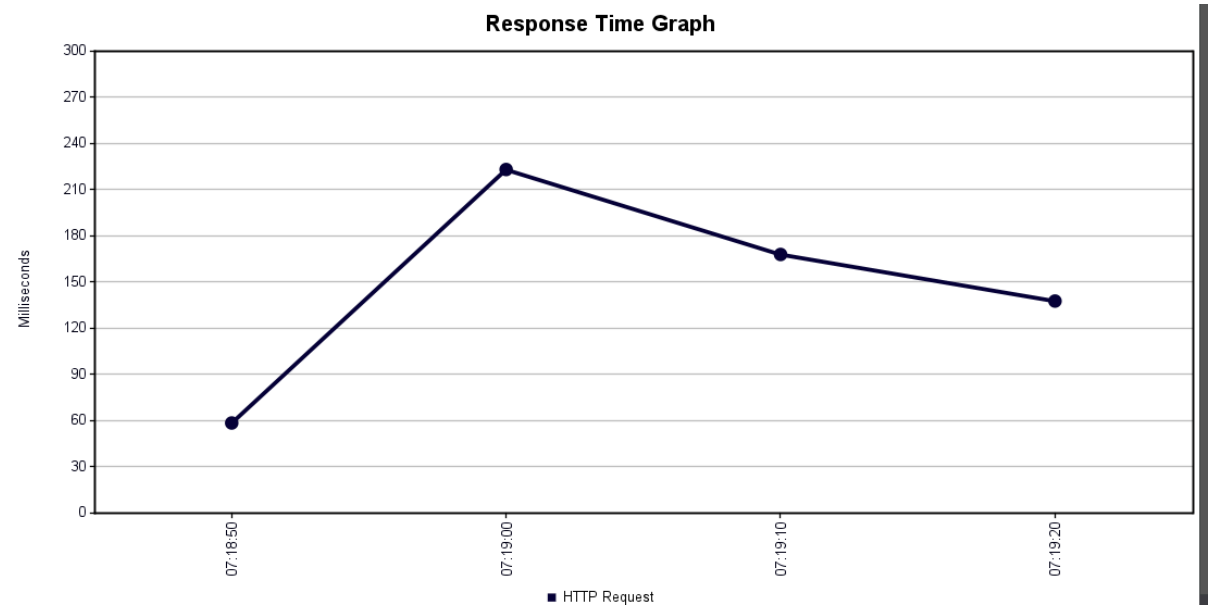


10. GET LICENSE PLATE

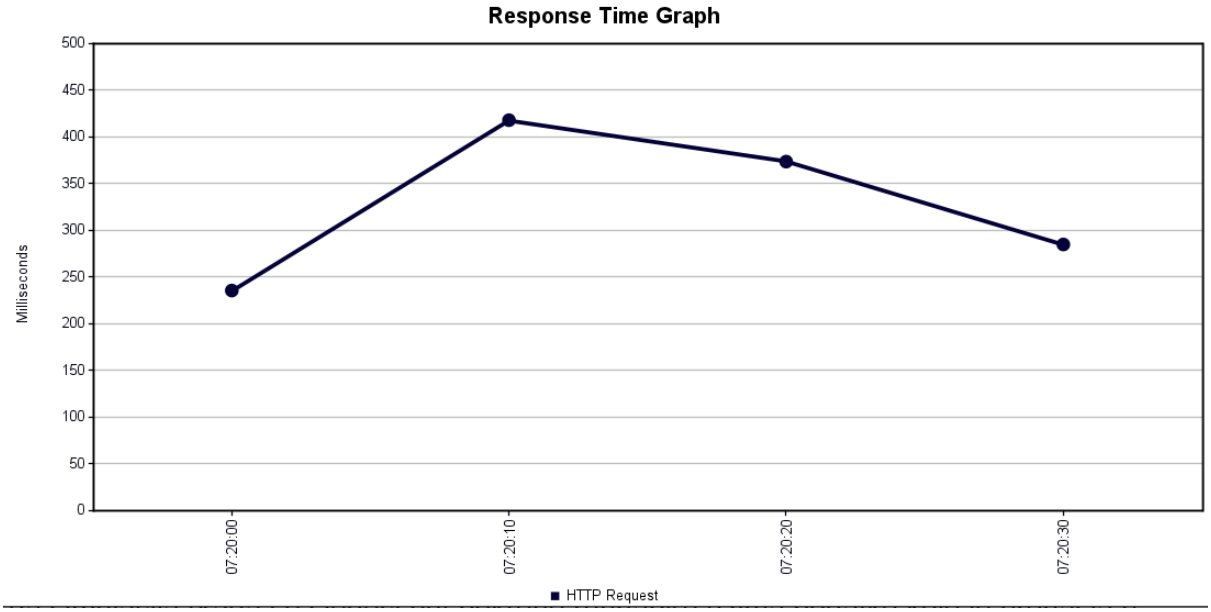
Posljednja funkcionalnost koju testiramo je dobavljanje privilegovanih tablica za određenu kapiju. Testiranje smo započeli sa 500 korisnika, koji su u 10 sekundi poslali po 50 zahtjeva. Testiranje je obavljeno u 3 iteracije.



1000 korisnika, po 50 zahtjeva. 2 iteracije.



2000 korisnika, po 20 zahtjeva. 2 iteracije



Zaključak ove sekcije testiranja je da je aplikacija pokazala prilično dobre performanse, posebno za rad sa bazama podataka. Upiti nad MySQL bazom su pokazali odlične performanse, zbog pisanja čistog sql koda (nismo koristili biblioteku koja je taj kod generisala za nas, tako da smo izbacili jednog posrednika). Influx baza je, zbog velike količine podataka, pokazala nešto lošije performanse, međutim ipak zadovoljavajuće. Optimizacija rada sa Influx bazom bi bilo prebacivanje svih upita na Flux jezik, tj. izbjegavanje dalje obrade podataka na serveru. Ovu optimizaciju smo izvršili na ~95% upita, a preostalih ~5% nam ostaje zbog slabijeg poznavanja jezika za upite ili ograničenja koje Flux jezik pruža, prilikom pisanja upita.

- **Testiranje performansi sistema usljed povećanja broja uređaja (simulatora) koji komuniciraju sa platformom**

Tokom razvoja aplikacije, pažljivo se vodilo računa kako i gdje se otvaraju thread-ovi. Pored simulacije, veći dio servera je paralelizovan. Potreba za paralelizacijom servera se prvi put pojavila prilikom upisa u Influx bazu. Upisuju se sve vrijednosti koje simulatori generišu. Problem na koji smo naišli je blokiranje servera, prilikom komunikacije sa simulatorima. To smo riješili ponovo nitima, ali smo tada trebali pažljivo voditi računa o informacijama koje se dijele između niti. Problem je uočen tek u kasnijim fazama razvoja, kada je i uspješno riješen. Ovo je rezultovalo zadovoljavajućim odzivom aplikacije za rad sa velikim brojem simulatora. U slučaju prebacivanja aplikacije u produkciju, ovo je nešto što bi zahtjevalo dodatnu optimizaciju.

LAMPA

	100 simulatora	1000 simulatora	2000 simulatora
on/off	3.76 ms	200.92 ms	5s
auto on/off	3.25 ms	192.14 ms	~ 4 s
graph data	20.24 ms	27s	~1min
kreiranje	8.72 ms	150.12 ms	2 - 3 min

KAPIJA ZA VOZILA

	100 simulatora	1000 simulatora	2000 simulatora
otvaranje/zatvaranje	2.89 ms	212.12 ms	7s
privatni/javni režim	4.11 ms	180.96 ms	11 s
graph data	45.12 ms	34 s	1 - 2 min
kreiranje	90 ms	244.65 ms	~ 4 min

SISTEM PRSKALICA

	100 simulatora	1000 simulatora	2000 simulatora
on/off	4.25 ms	250.47 ms	11 s
dodavanje režima	4.89 ms	620.14 ms	1 - 2 min
graph data	22.14 ms	881.13 ms	2 - 3 min
kreiranje	12 ms	420.17 ms	2 - 3 min