Sva testiranja su obavljena na racunaru koji ima Intel Core i7-10750H procesor sa 6 jezgara. Racunar posjeduje 16GB RAM memorije. Testiranje je vrseno uz pomoc JMeterMeter open-source alat za testiranje performansi i funkcionalnosti softvera.

## Get Lamp Illumination History

Sledeci grafici predstavljaju prikaz vremena odziva sistema za funkcionalnost dobavljanja istorije upisanih vrijednosti za osvjetljenje koje upisuje pametni uredjaja “Lampa” u InfluxDB. Sva testiranje za Get Illumination History funkcionalnost su pokrenuta dok je istovremeno simuliran rad 50 pametnih uredjaja. Vremenski opseg za koji su generisani podaci je 3 dana.

Primjeceno je da je sistem izuzetno spor kada postoji jako velika kolicina podataka (na primer, kada imamo generisane podatke za celi mesec, gde generisanje podataka se desava na svakih 45 sekundi). Iz tog razloga uvedena je agregacija za InfluxDB dobavljanje vrijednosti.

Konkretno, u ovoj funkcionalnosti, u zavisnosti od vremena bira se razlicito vrijeme agregiranja (za krace vremense periode, agregacija je na “3m”, dok kod velikih vremenskih perioda agregacija je “30m”. Ovo je dovelo do znacajnih poboljasnja preformansi. Takodje, korisnik vise ne mora cekati da se na front dijelu aplikacije ucitava grafik ocitanih vrijednosti osvetljenja.

Prikaz funkcije za agregiranje koja je pomogla u rjesavanju problema:

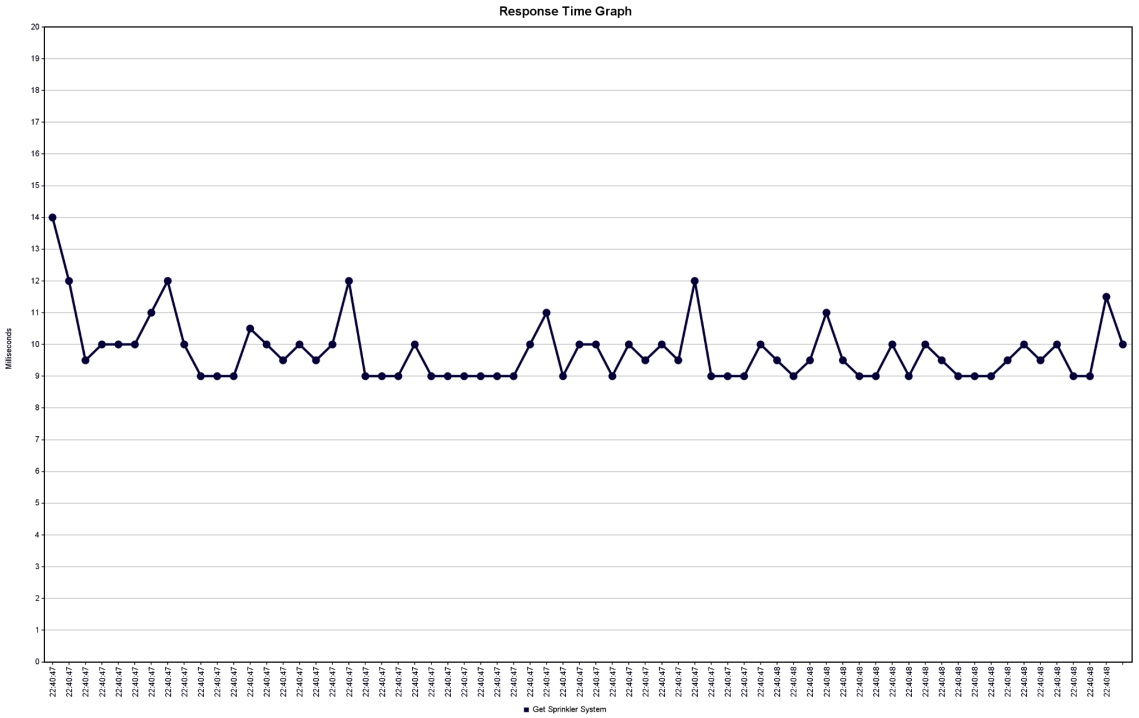
```
from(bucket: "measurements")
      |> range(start: 13263472, stop: 676241729)
      |> filter(fn: (r) => r["_measurement"] == "lam-device")
      |> filter(fn: (r) => r["_field"] == "illumination")
      |> filter(fn: (r) => r["id"] == "5")
      |> aggregateWindow(every: “30m”, fn: “mean”, createEmpty: false)
      |> sort(columns: ["_time"], desc: false)
```

Funkcija koja odredjuje velicinu aggregateWindow-a:

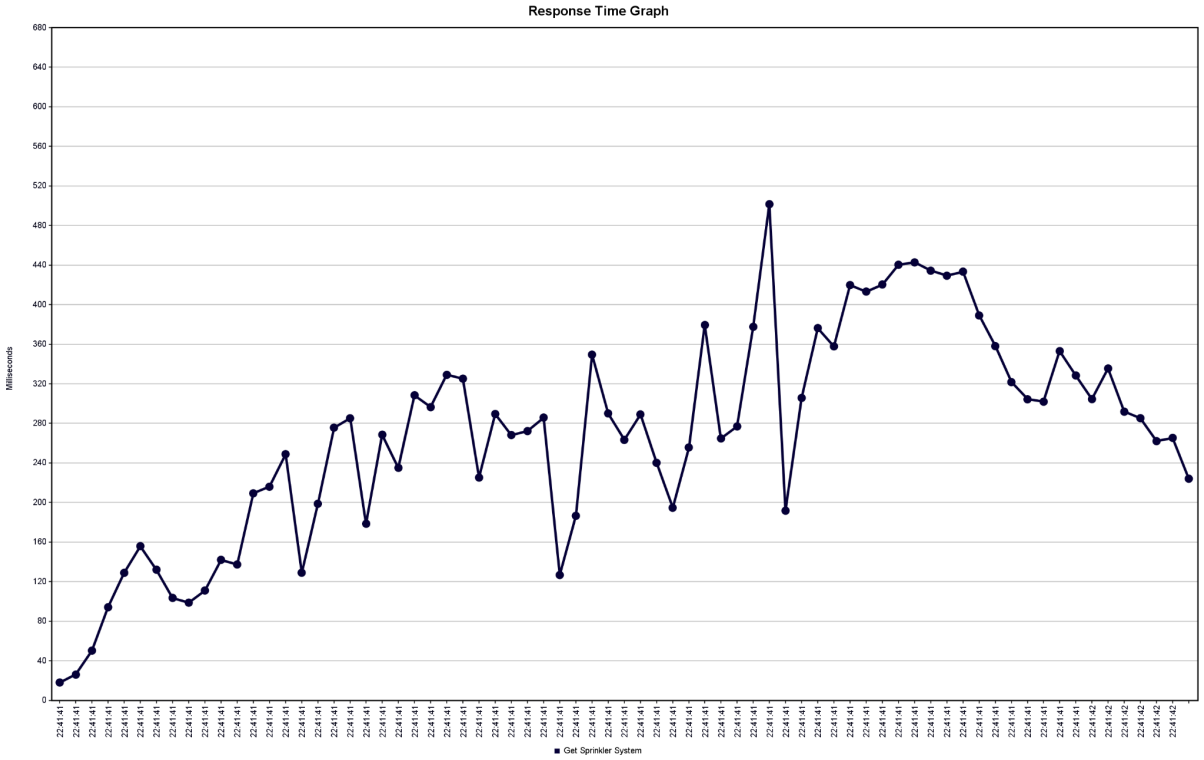
```
public static String getAggregationParam(LocalDateTime startTime,
LocalDateTime endTime) {
   Duration duration = Duration.between(startTime, endTime);
   if (duration.toHours() <= 1) return null;
   if (duration.toHours() <= 12) return "3m";
   else if (duration.toDays() < 5)  return "7m";
   else if (duration.toDays() < 10) return "15m";
   else if (duration.toDays() < 15) return "20m";
   else if (duration.toDays() < 20) return "25m";
   else if (duration.toDays() < 31) return "30m";
   else return null;
}
```

Trenutne preformanse su prikazane u sledecim grafovima:

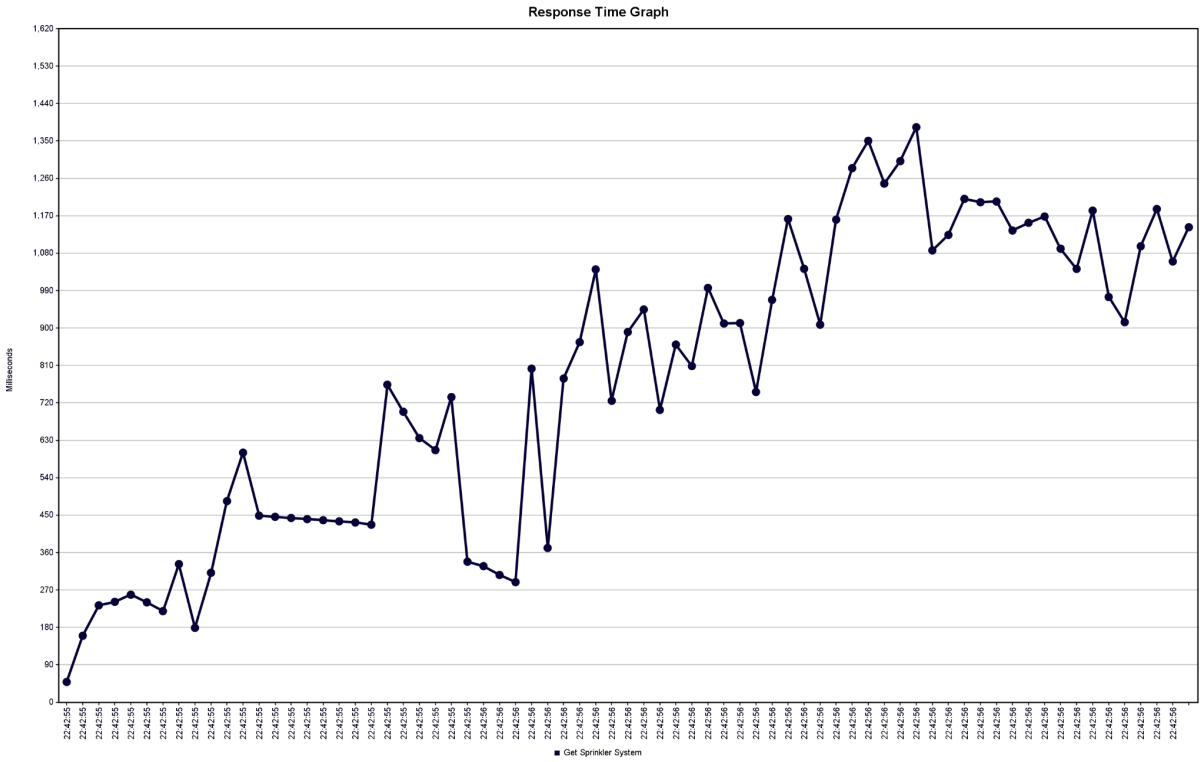
Broj thread-ova (user-a): 100



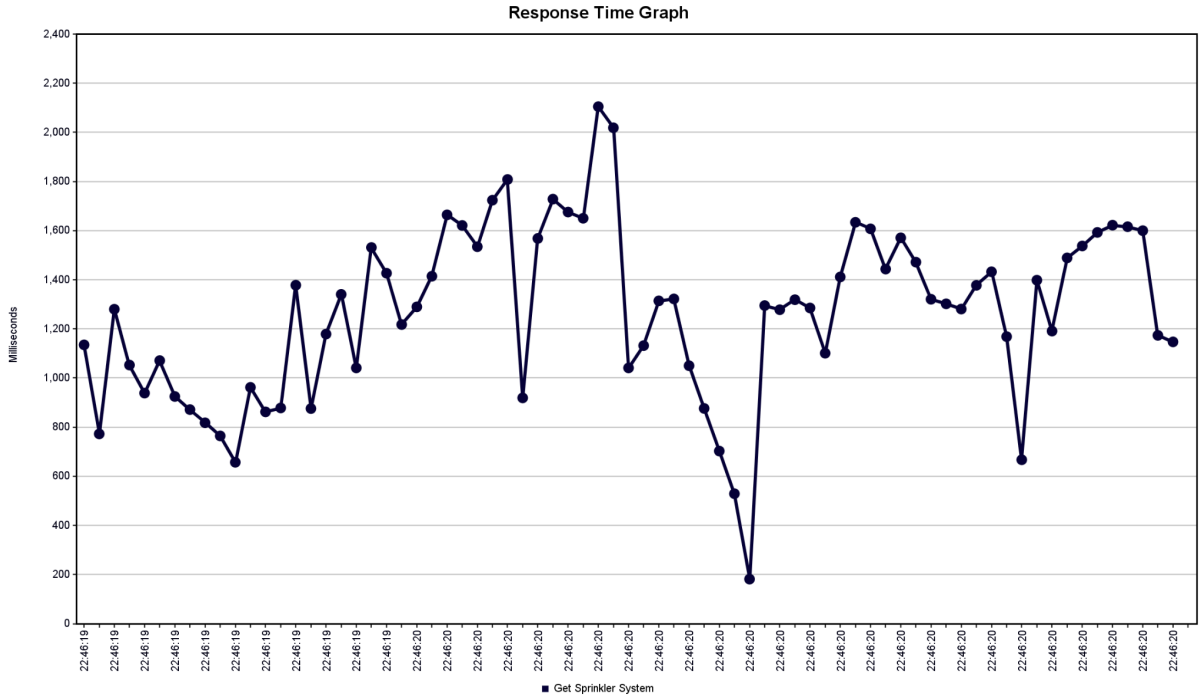

Broj thread-ova (user-a): 500



Broj thread-ova (user-a): 1000




Broj thread-ova (user-a): 1300




Svakako, mozemo primetiti da sa povecanjem broja usera povecava se i vreme odziva.

# Get Online-Offline (Status) History

Sledeci grafici predstavljaju prikaz vremena odziva sistema za funkcionalnost dobavljanja istorije upisanih vrijednosti za heartbeat koje svaki tip pametnih uredjaja u InfluxDB. Sva testiranje za Get Online-Offline Historyfunkcionalnost su pokrenuta dok je istovremeno simuliran rad 50 pametnih uredjaja. Vremenski opseg za koji su generisani podaci je 3 dana.

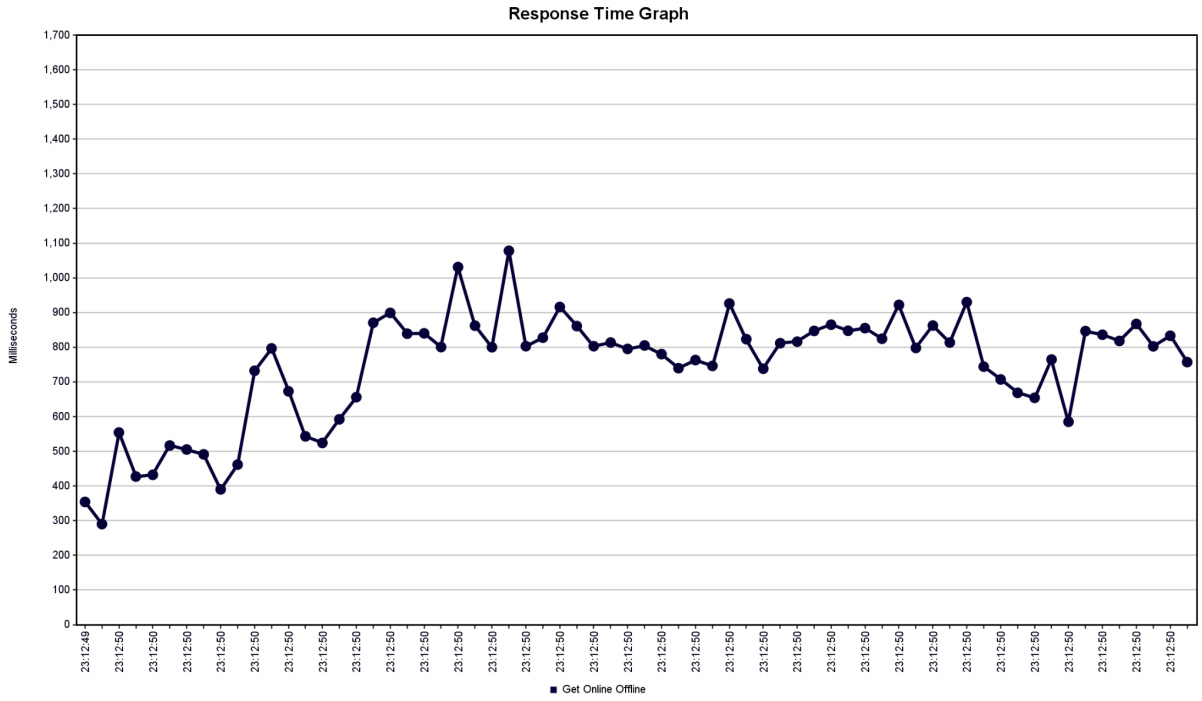
Kao i u prethodnom primjeru testiranja, zbog generisanja velikog broja heartbeat vrijednosti, postoji problem u dobavljanju i prikazivanju samih vrijednosti za heartbeat, jer je potrebno da se mnogo njih obradi. Stoga, iskoriscena je agregacija kao rjesenje.

U ovom slucaju, posto je haertbeat osjetljivija informacija od vrednosti osvetljenja koje je lampa ocitala, obavljena je manja agregacija. Funkcija koja odredjuje parametar aggationWindow za ovaj slucaj je:

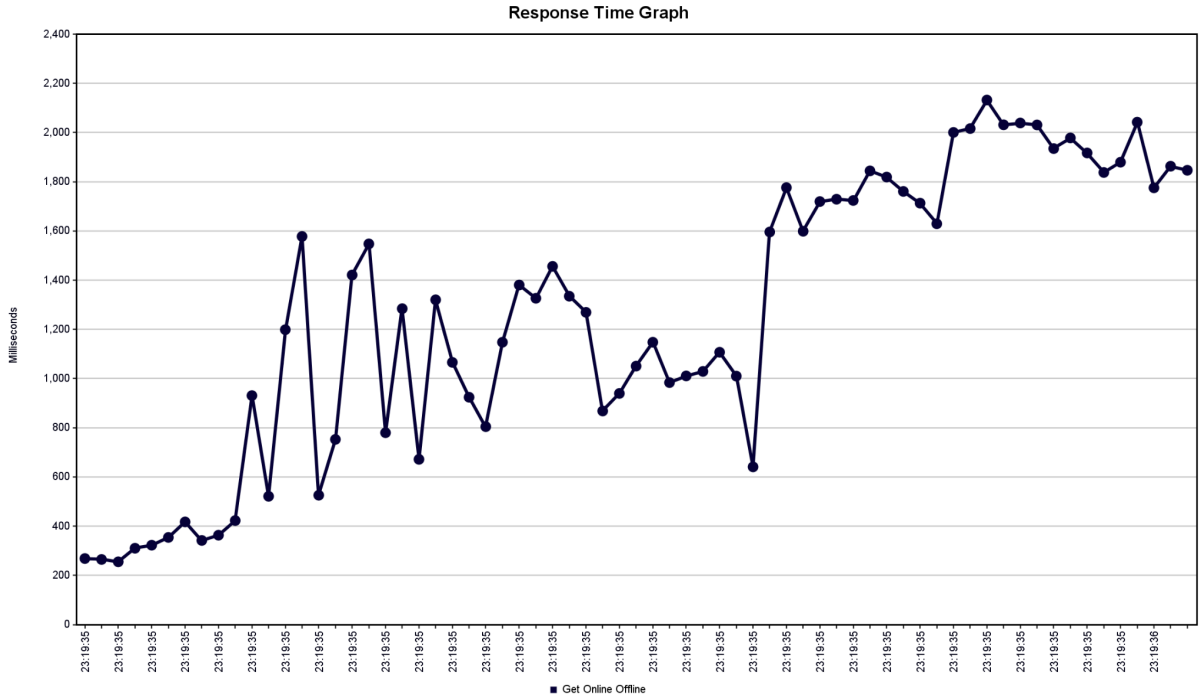
```
public String getAggregationParam(LocalDateTime startTime, endTime) {
        Duration duration = Duration.between(startTime, endTime);
        if (duration.toDays() <= 3) return null;
        if (duration.toDays() <= 10)return "5m";
        else if (duration.toDays() < 20) return "10m";
        else if (duration.toDays() <= 31)return "15m";
        else return null;
```

Zbog manje agregacije, i vrijeme odziva je vece. Ovo bi se moglo rijesiti vecom agregacijom, kao i u prethodnom primjeru testova. Jos jedno rjesenje, jeste da platforma periodicno (na primer na pocetku svakog mseca) pokrene logiku koja bi vrsila smanjivnje broja vrednosti koje su unete u InfluxDB.

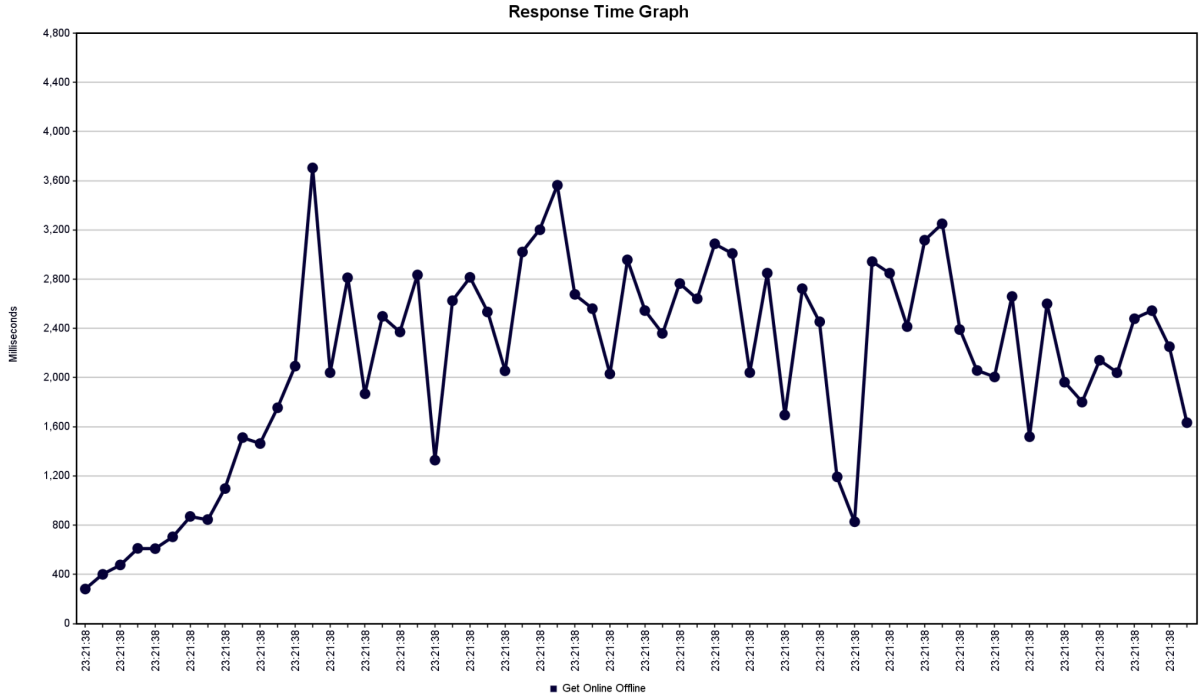
Trenutne preformanse su prikazane u sledecim grafovima:
Broj thread-ova (user-a): 100



Broj thread-ova (user-a): 400




Broj thread-ova (user-a): 700



# Get Vehicle Plate History

Sledeci grafici predstavljaju prikaz vremena odziva sistema za funkcionalnost dobavljanje o podataka o prisutnosti odredjenog vozila unutar dvorista tokom odredjenog vremena. Sva testiranje za Accept Property Request funkcionalnost su pokrenuta dok je istovremeno simuliran rad 100 pametnih uredjaja.

U pocetku, pretraga podataka za InfluxDB je radjena uz pomoc imena polja (field). Medjutim, tokom istrazivanja, vidjela sam da je bolje pretrazivati po tagovima i koristiti tagove za filtriranje i slanje upita na InfluxDB.

Prvi razlog jeste, sto uz pomoc tagova moze da se vrsi filtriranje po vise kriterijuma. Konkretno za ovaj primer, filtriranje se moze vrsiti po parametru mode (javan ili privatan) i po tome koje izazvao neku akciju (u ovom slucaju registarska tablica, korisnik ili sistem).

Drugi razog koriscenja tagova jeste sama implementacija u InfluxDB. Tagovi u InfluxDB se koriste za indeksiranje podataka. Pretraga po tagovima  je brza jer InfluxDB koristi indekse tagova za direktan pristup relevantnim podacima, a kod polja (field) ne koristi se indeksiranje.

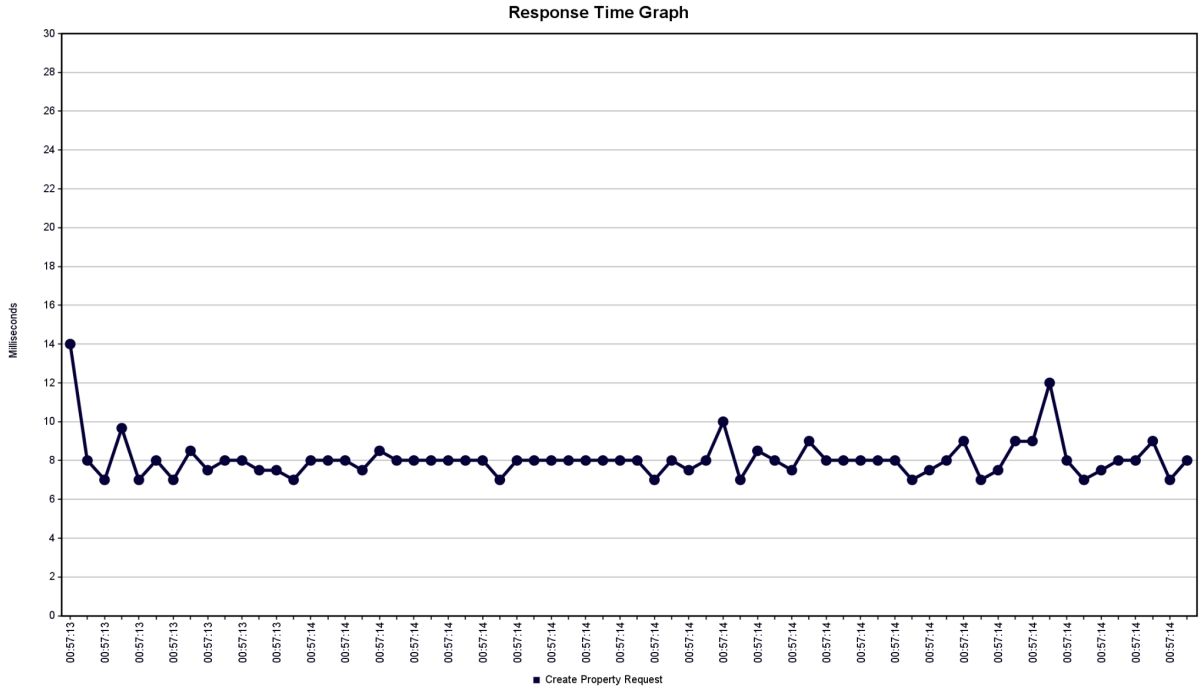
Primer InfluxDB upita sa filtriranjem po tagovima:

```
from(bucket: "measurements")
      |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
      |> filter(fn: (r) => r["_measurement"] == "vehicle-gate")
      |> filter(fn: (r) => r["_field"] == "vehicle-gate-action")
      |> filter(fn: (r) => r["id"] == "3")
      |> filter(fn: (r) => r["triggered_by"] == "BBB-2-BBB")
```

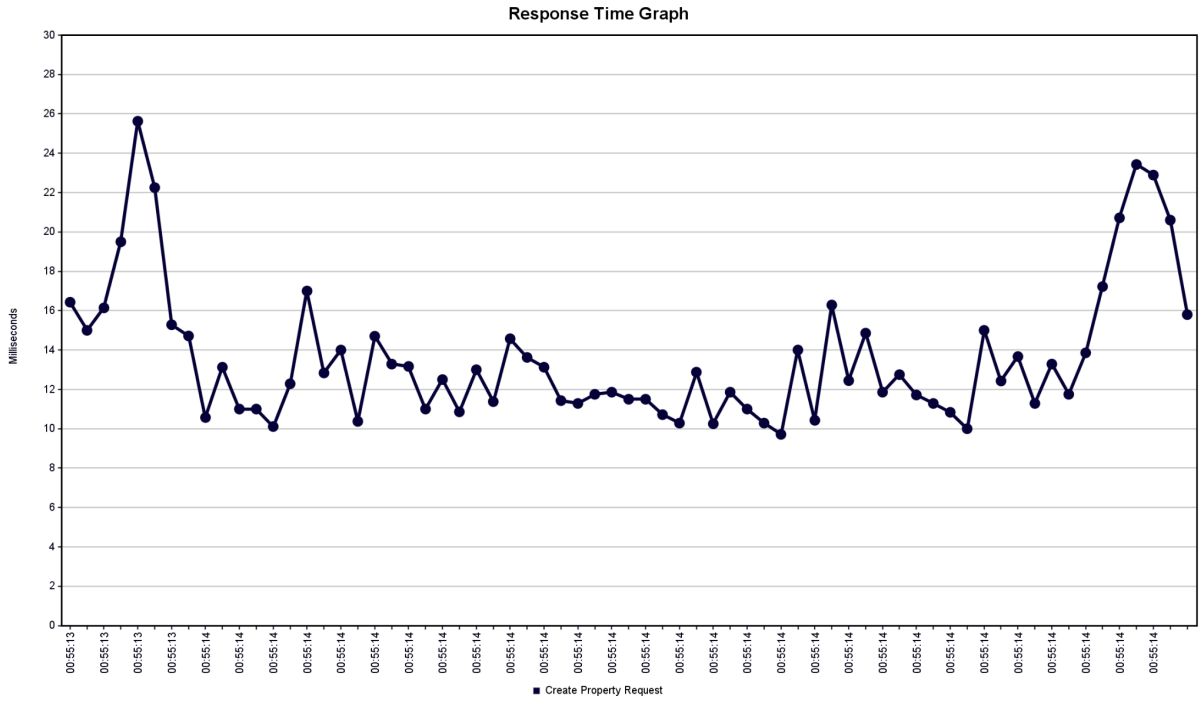
Sama organizacija podataka u InfluxDB je prilagodjen bas ovakvim upitima. Stvari po kojima ce biti moguca pretraga identitea (ko je izazvao neku akcuju (triggered by) i koji je mode ukljucen (private ili public)) su upsani kao tagovi, te je omoguceno lase i brze filtriranje upita koji ce da se prikazu korisniku kao odgovor.

Trenutne preformanse su prikazane u sledecim grafovima:

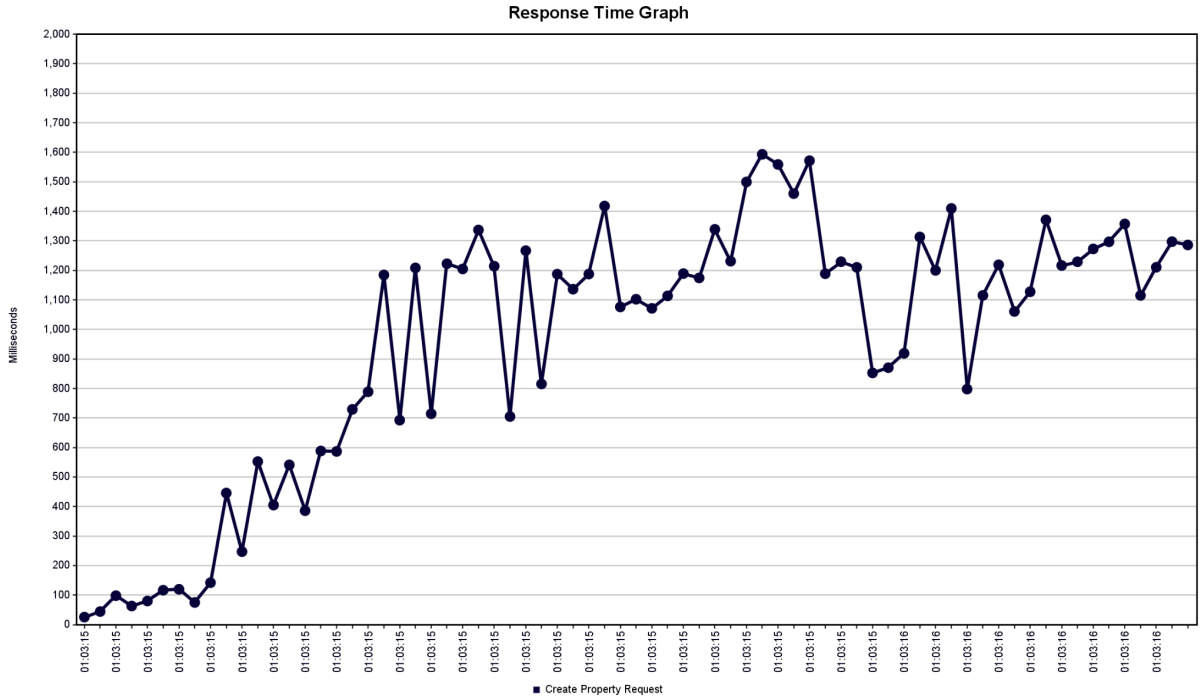
Broj thread-ova (user-a): 100



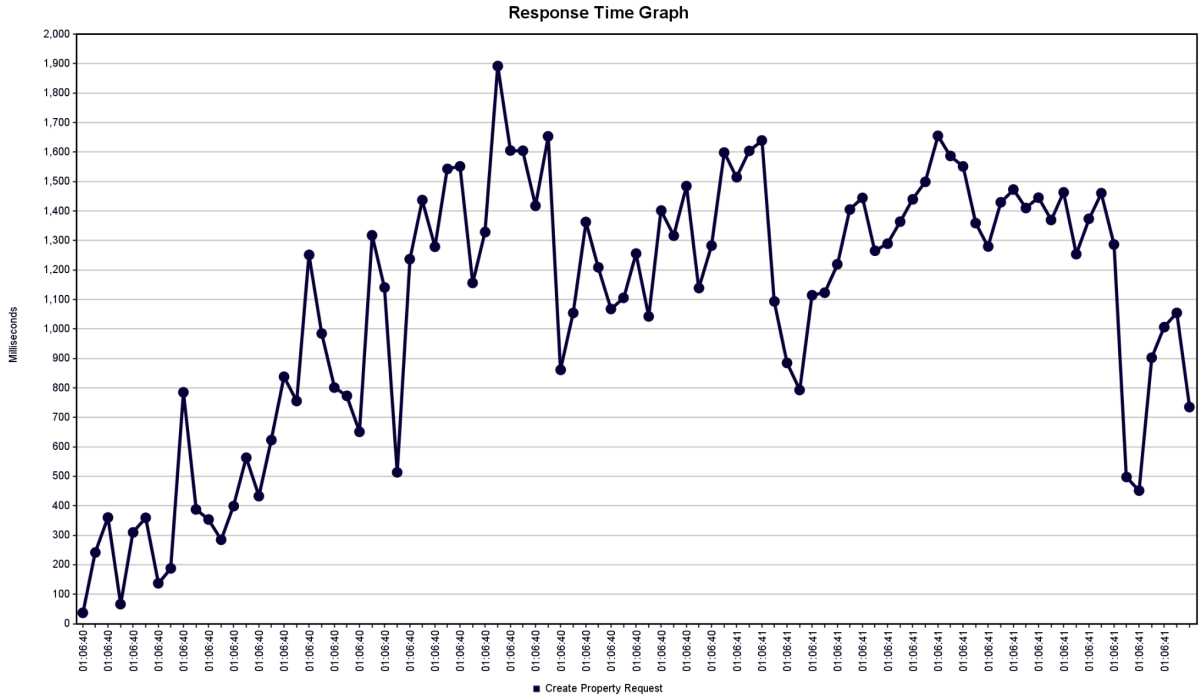

Broj thread-ova (user-a): 500



Broj thread-ova (user-a): 1000




Broj thread-ova (user-a): 1300



# Get Vehicle Gate Actions History

Sledeci grafici predstavljaju prikaz vremena odziva sistema za funkcionalnost dobavljanje o podataka o svim akcijama u nekom vremenskom periodu za pametni uredjaj “Kapija”. Sva testiranje za Get Vehicle Gate Actions History funkcionalnost su pokrenuta dok je istovremeno simuliran rad 100 pametnih uredjaja.

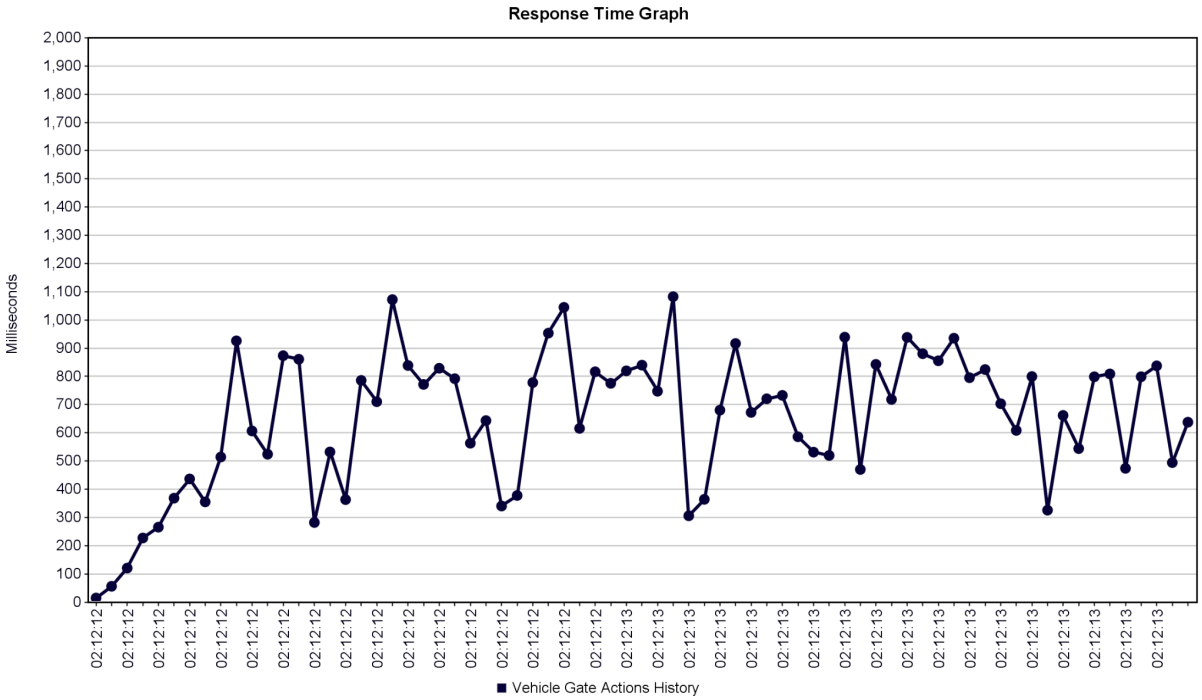
Tokom ovog testa cilj je bio prikazati kako se sistem ponasa kada treba da dobavi razlicite kolicine podataka sa istim skupom korisnika. Konkretno, broj user-a odnosno thread-ova je 1000, ali se mijenja vremenski opseg za koji s traze podaci.

Svaki sledeci test prikazuje ponasanje, odnosno odiv sistema kada se povecava vremenski opseg za nekoliko dana. Povecanjem opsega povecava se i kolicina podataka za dobavljanje. Parametri za filtriranje su uvijek isti (nema ih da bi se dobavila sto veca kolicina podataka)

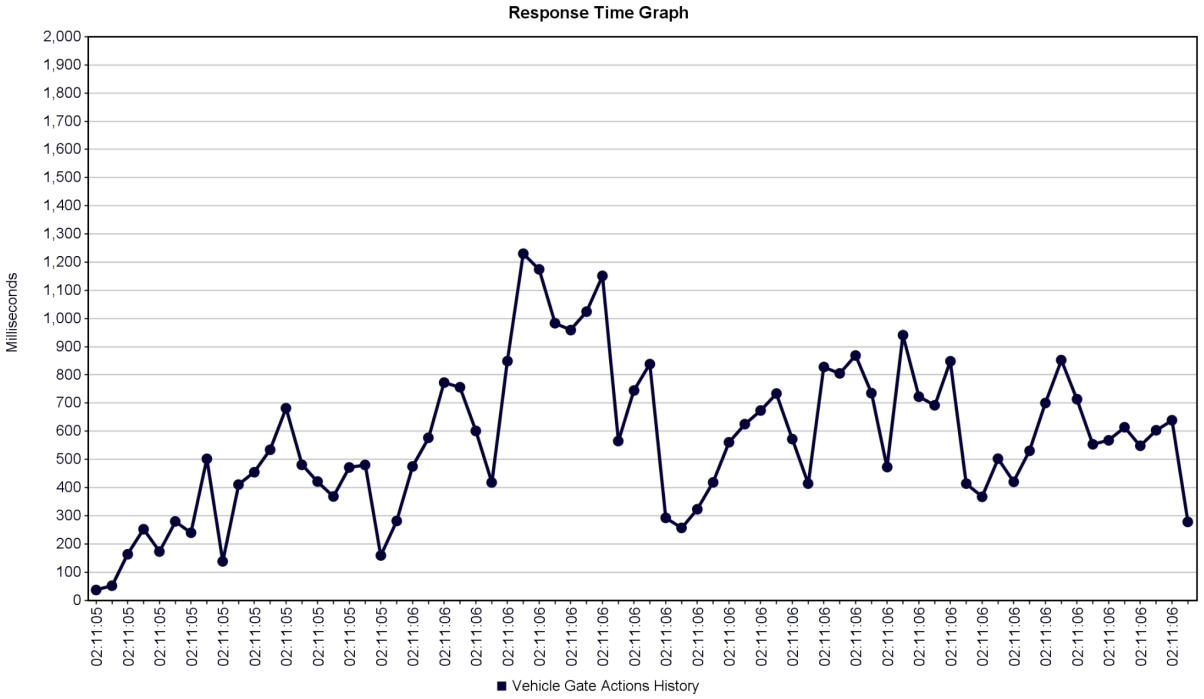
Mozemo primjetiti da se odziv povecao sa povecanjem vremenskog perioda, ali ne mnogo (maksimalna vrijednost za 1 dan je 1100ms, za 15 dana 1200ms, a za 30 dana 1400ms)

Trenutne preformanse su prikazane u sledecim grafovima:

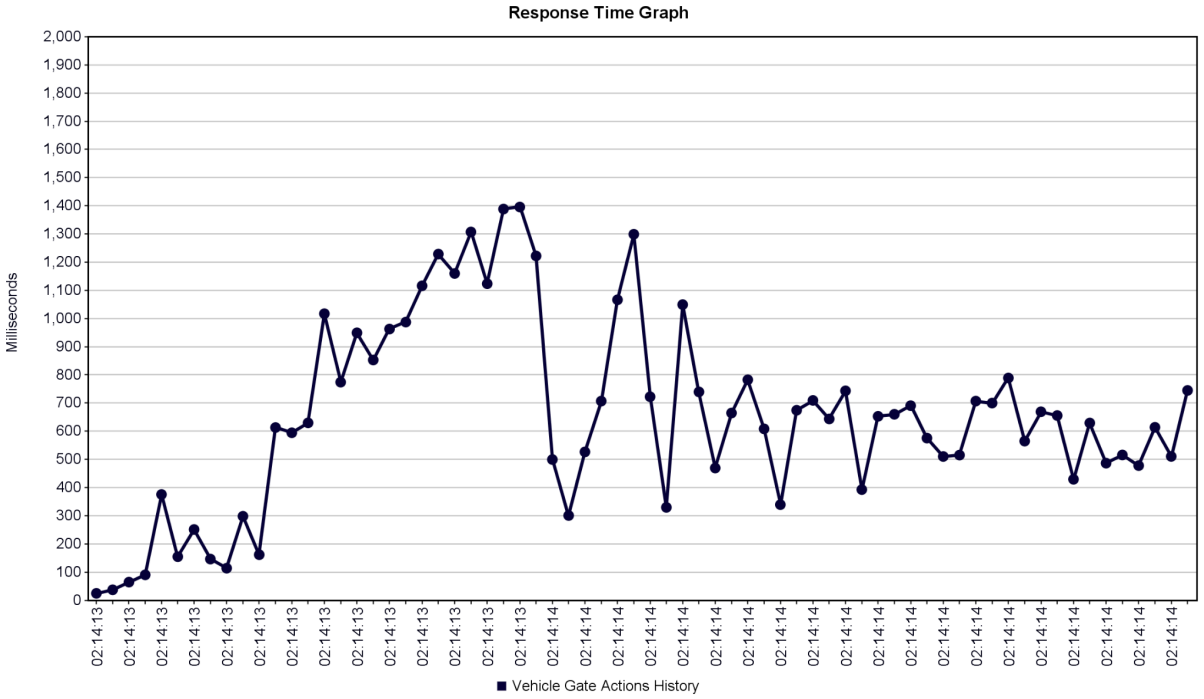
Broj thread-ova (user-a): 1000
Broj dana u vremenskom opsegu: 1



Broj thread-ova (user-a): 1000
Broj dana u vremenskom opsegu: 15




Broj thread-ova (user-a): 1000
Broj dana u vremenskom opsegu: 30



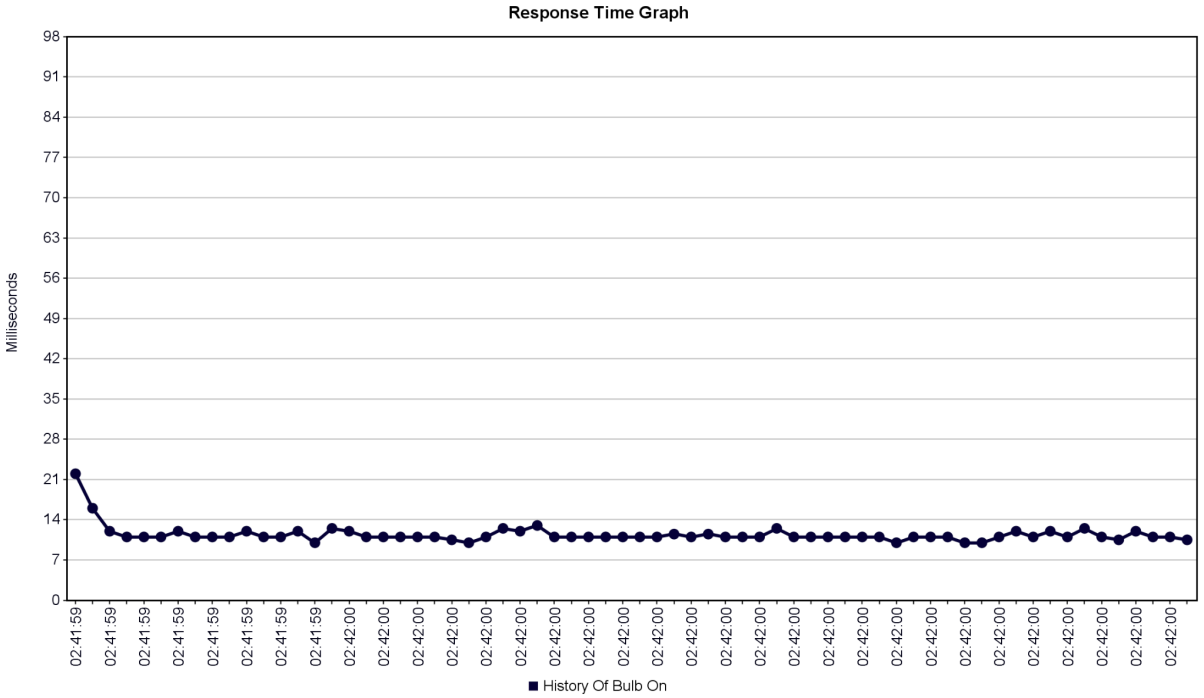
# Get Bulb On History

Sledeci grafici predstavljaju prikaz vremena odziva sistema za funkcionalnost dobavljanje o podataka o trenucima kada jeste/nije bila upaljena sijalica za pametni uredjaj “Lampa”. Sva testiranje za Get Bulb On History funkcionalnost su pokrenuta dok je istovremeno simuliran rad 100 pametnih uredjaja.

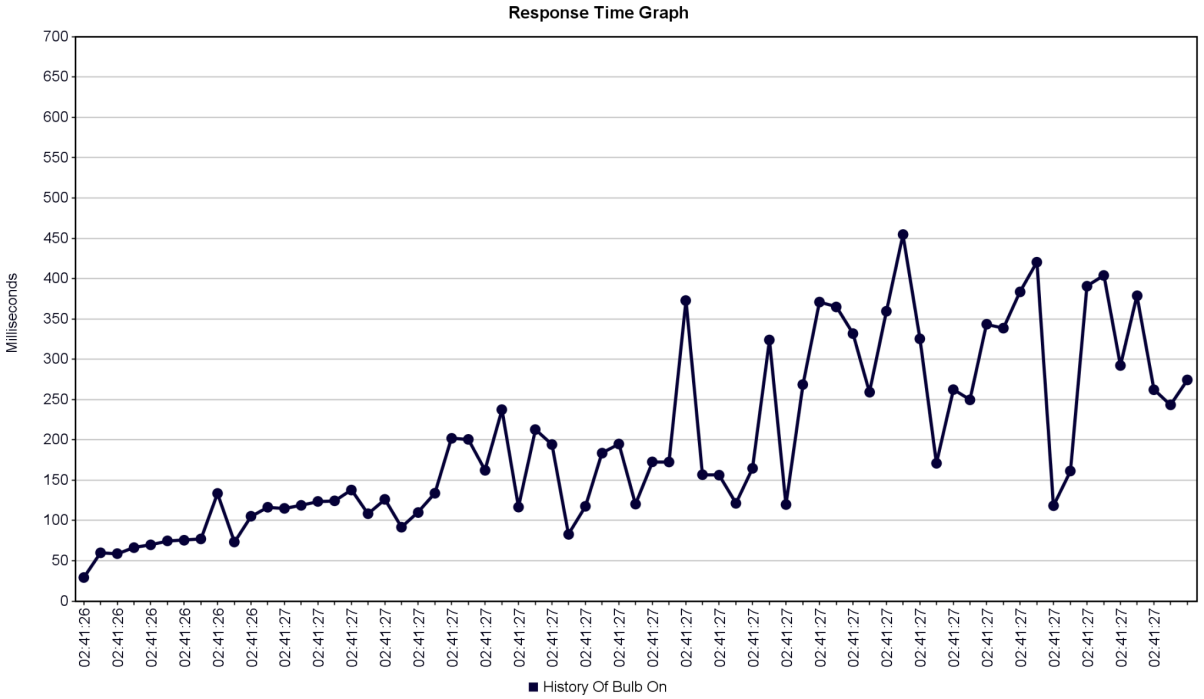
U pocetku implementacije, svaki put uz ocitanu vrijednost osverljenja, lampa je vracala i vrijednost tipa bool, koja se odnosila na to da li je u tom trenutku sijalica pametnog uredjaja upaljena ili nije.

Medjutim, naknadno sam dodala tabelu akcija, u kojoj je jasno definisano bilo i kada se sijalica pametnog uredjaja pali (sijalicu moze upaliti korisnik ili sistem). Posto sam vec imala te podatke za drugu funkcionalnost, da ne bih imala redudantne podatke, odlucila sam koristiti tabelu akcija za grafik ukljuceno/iskljuceno svjetlo, cime sam smanjila broj podataka za upis i takodje broj podataka za citanje pri konstrukciji grafa (manje se akcija belezi nego da lampa na svakih 45 sekundi upisuje vrednost tipa boolean true/false za ukljucen/iskljucen.

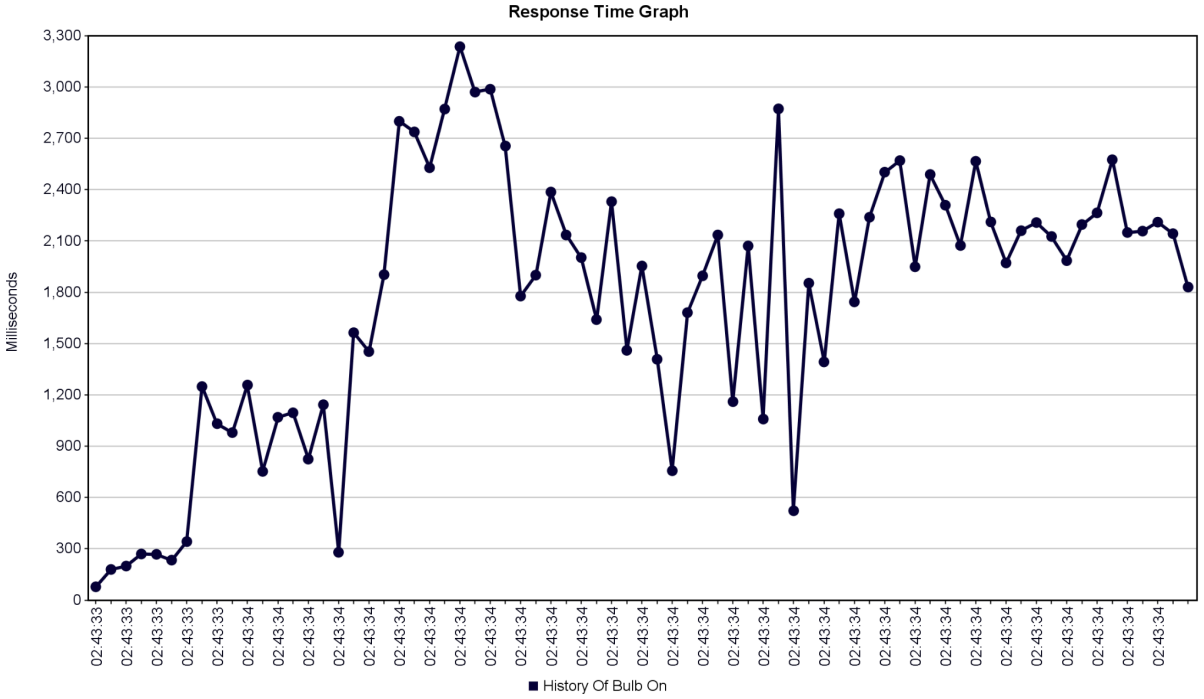
Trenutne preformanse su prikazane u sledecim grafovima:
Broj thread-ova (user-a): 100



Broj thread-ova (user-a): 500




Broj thread-ova (user-a): 1300



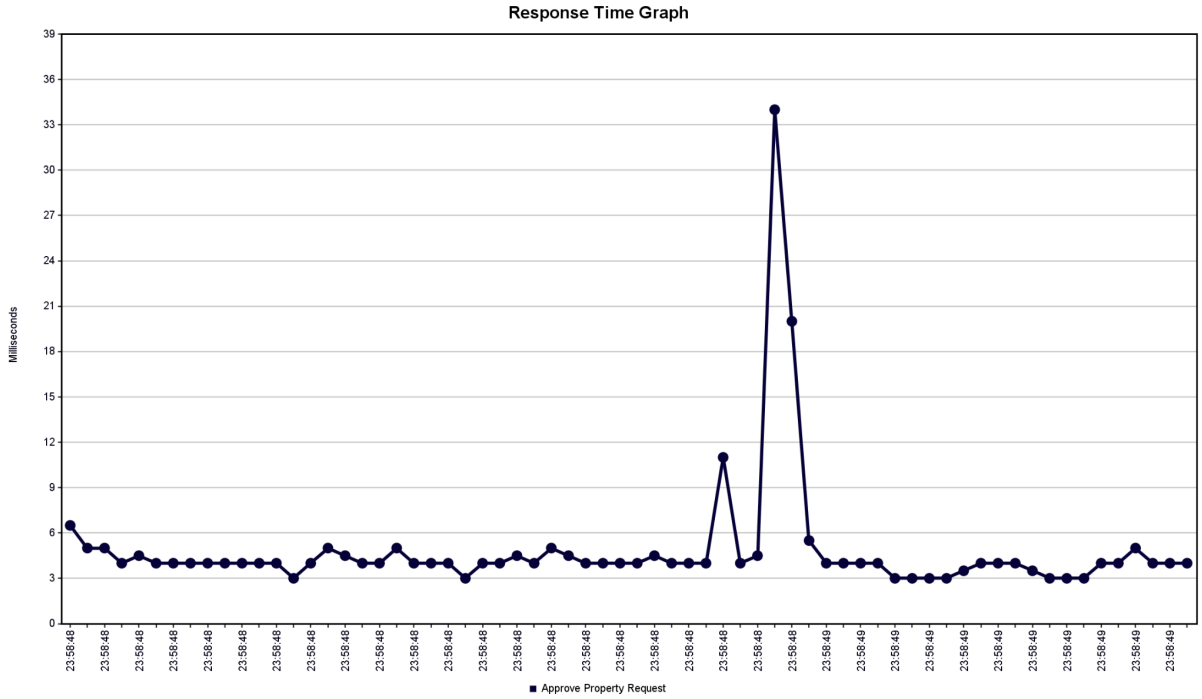
# Accept Property Request

Sledeci grafici predstavljaju prikaz vremena odziva sistema za funkcionalnost odobravanje/odbijanje zahtjeva za novu nekretninu. Sva testiranje za Accept Property Request funkcionalnost su pokrenuta dok je istovremeno simuliran rad 50 pametnih uredjaja.

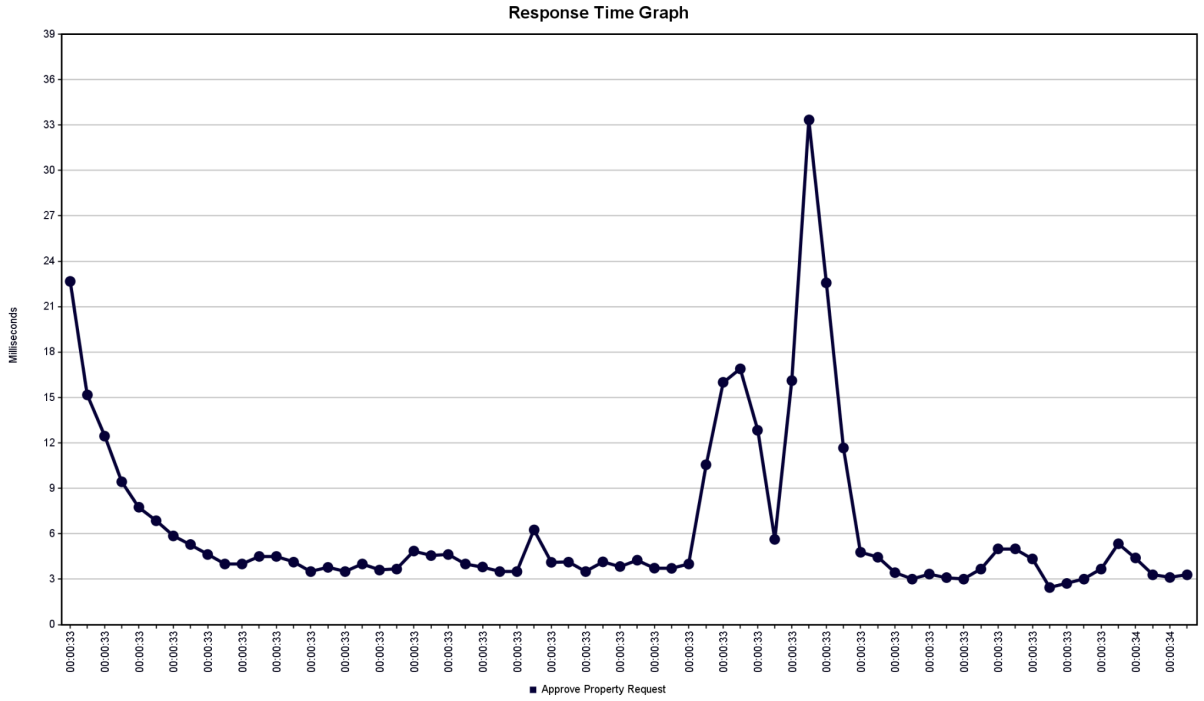
Problem do kojeg je doslo pri implementaciji jeste slanje email poruka, gdje je inicijalna verzija koda bila da se ceka dok se email ne posalje korisniku. Medjutim, ovo je trajalo dugo vremena, pa je stoga dodato asinhrono slanje email poruka, sto je smanjilo vrijeme samog odziva, jer se email poruka salje “u pozadini”.

Trenutne preformanse su prikazane u sledecim grafovima:

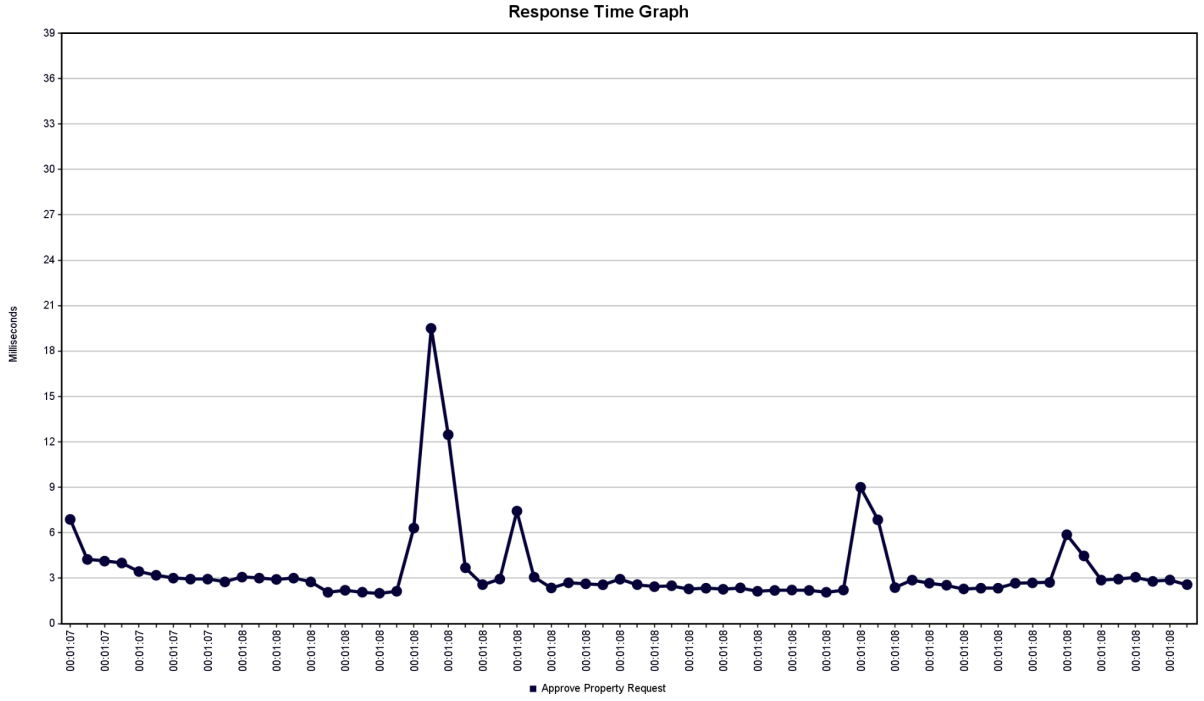
Broj thread-ova (user-a): 100



Broj thread-ova (user-a): 500




Broj thread-ova (user-a): 1000

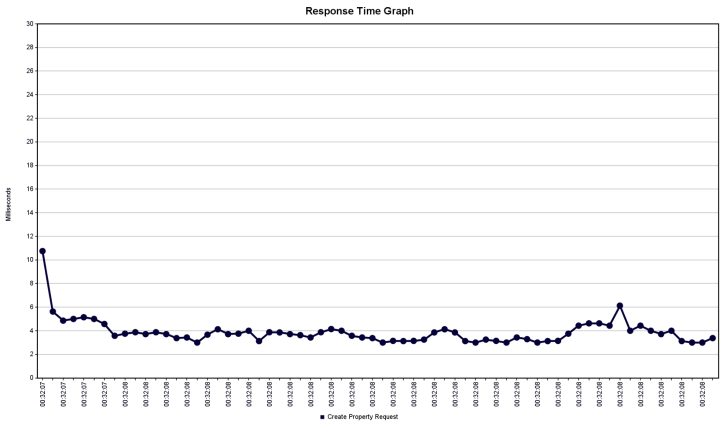


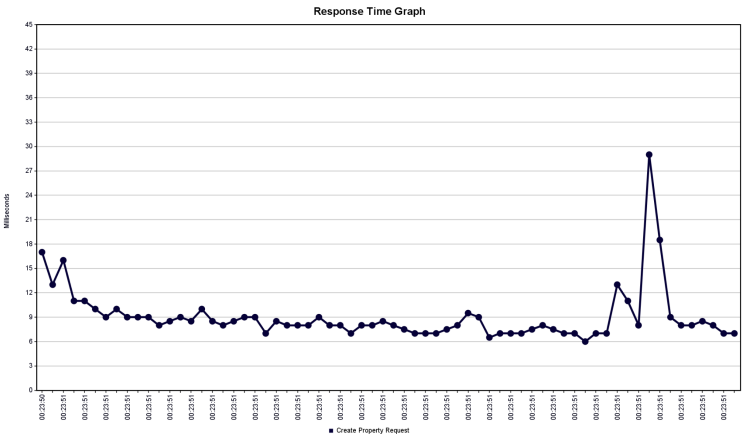
# Create (Deny) Property Request

Sledeci grafici predstavljaju prikaz vremena odziva sistema za funkcionalnost kreiranje zahtjeva za novu nekretninu. Sva testiranje za Accept (Deny) Property Request funkcionalnost su pokrenuta dok je istovremeno simuliran rad 50 pametnih uredjaja, odnosno 100 uredjaja radi poredjenja ponasanja sistema tokom rada vise simulacija.

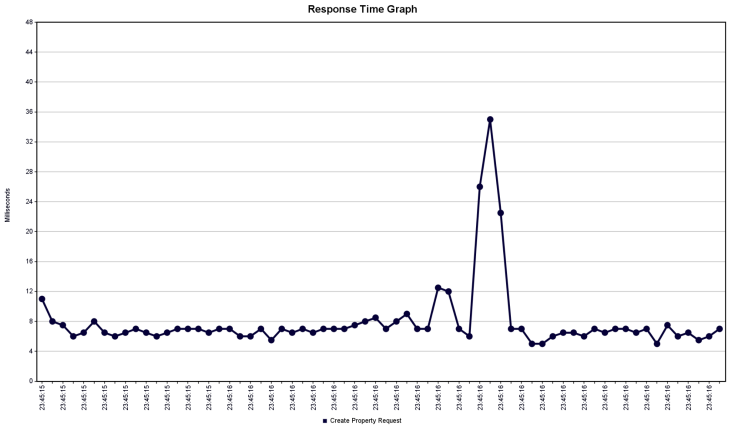
Iz grafova mozemo primjetiti da sa povecanjem simulacija dolazi i do malog povecaja odziva za funkcionalnost Create Property Request. Medjutim, na 1300 usera vreme odziva niti u jednom slucaju (sa vecim ili manjim opterecenjem) nije vece od 300ms.

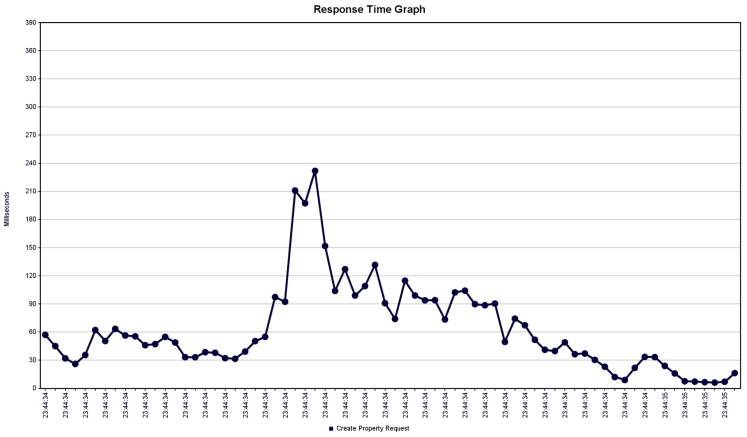
Broj thread-ova (user-a): 1000
[slika levo 50 simulacija ukljuceno, slika desno 100 simulacija ukljuceno]



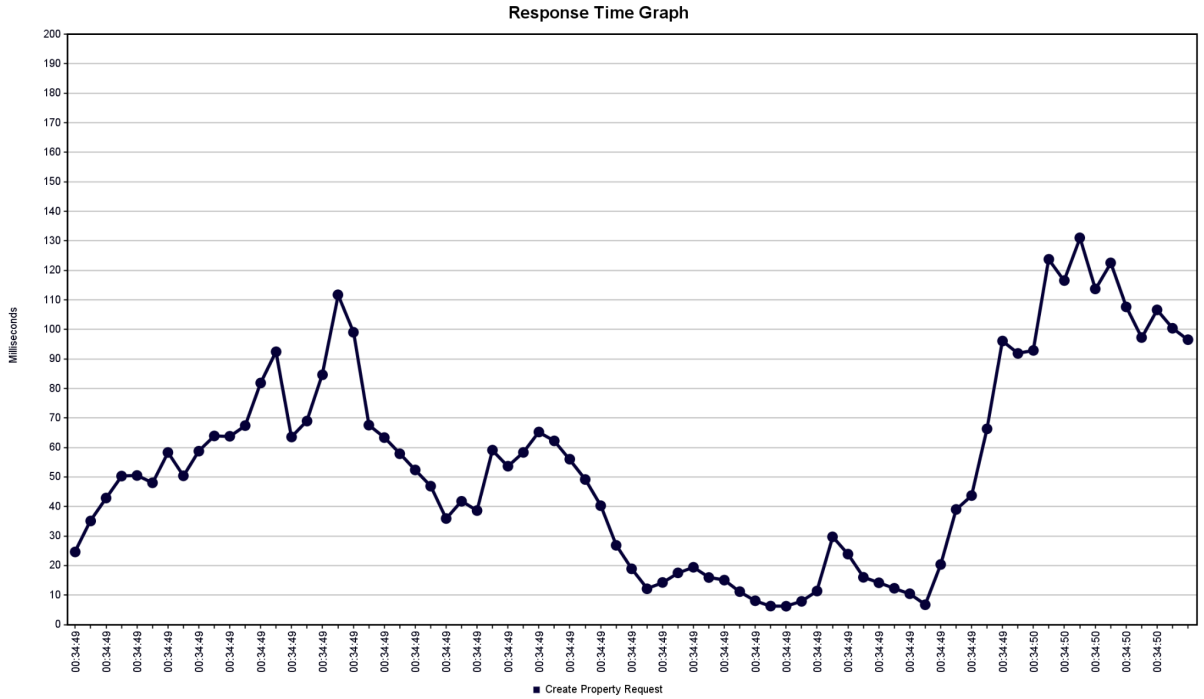




Broj thread-ova (user-a): 500
[slika levo 50 simulacija ukljuceno, slika desno 100 simulacija ukljuceno]

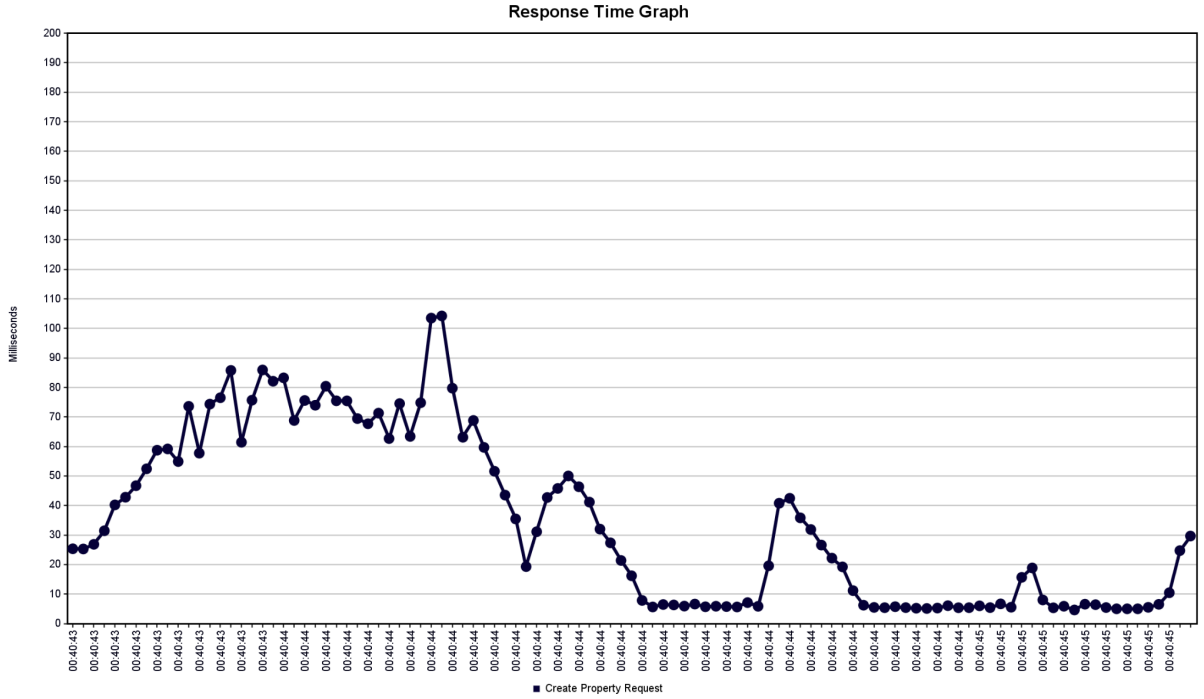


Broj thread-ova (user-a): 1000 [100 simulacija ukljuceno]



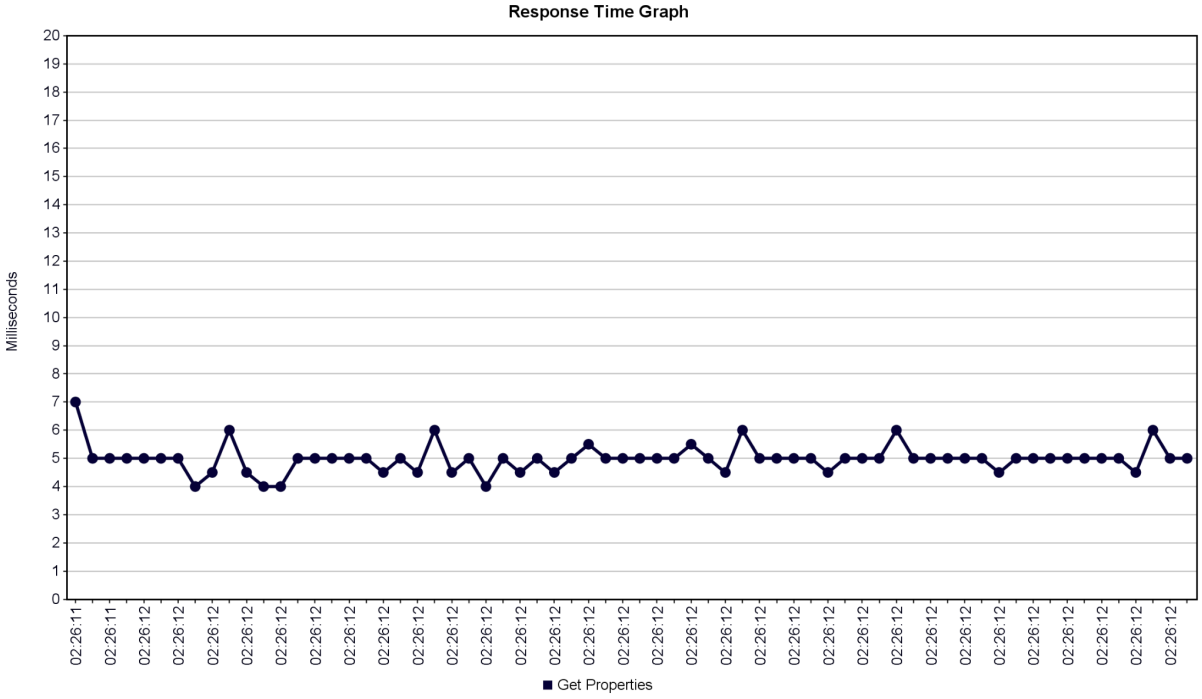

Broj thread-ova (user-a): 1300 [100 simulacija ukljuceno]



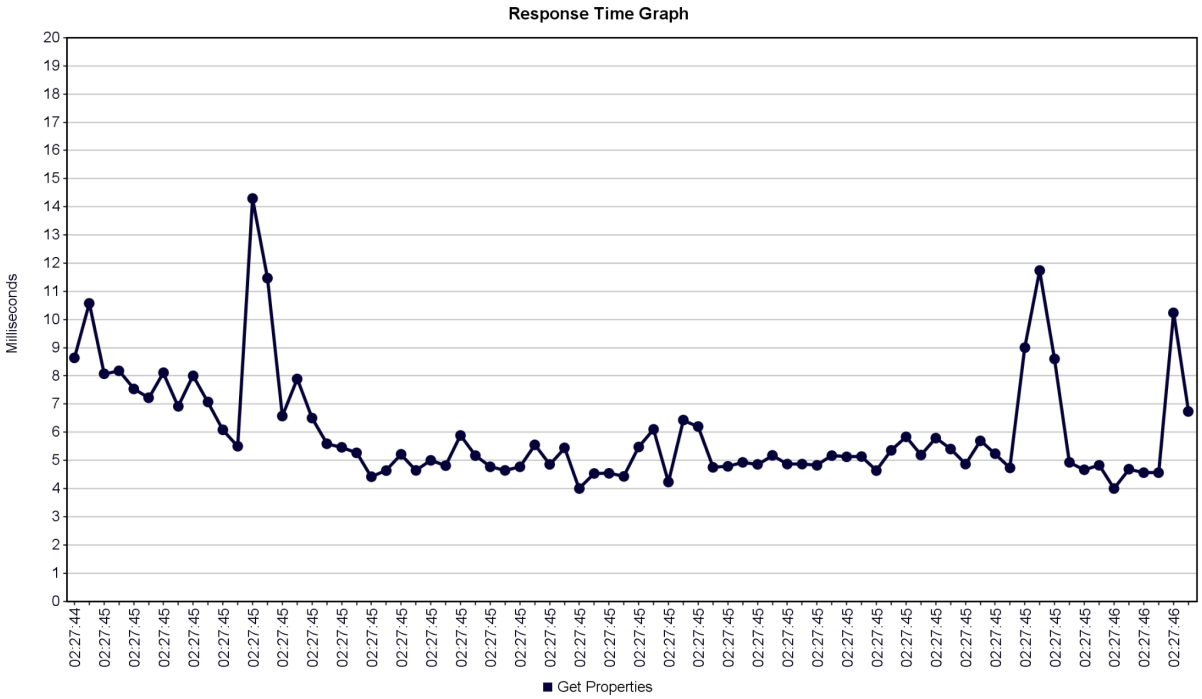
# Get Properties For One User

Prikaz vremena odziva sistema za funkcionalnost dobavljanja nekretnie. Sva testiranje za Get Properties By User Email funkcionalnost su pokrenuta dok je istovremeno simuliran rad 100 pametnih uredjaja.

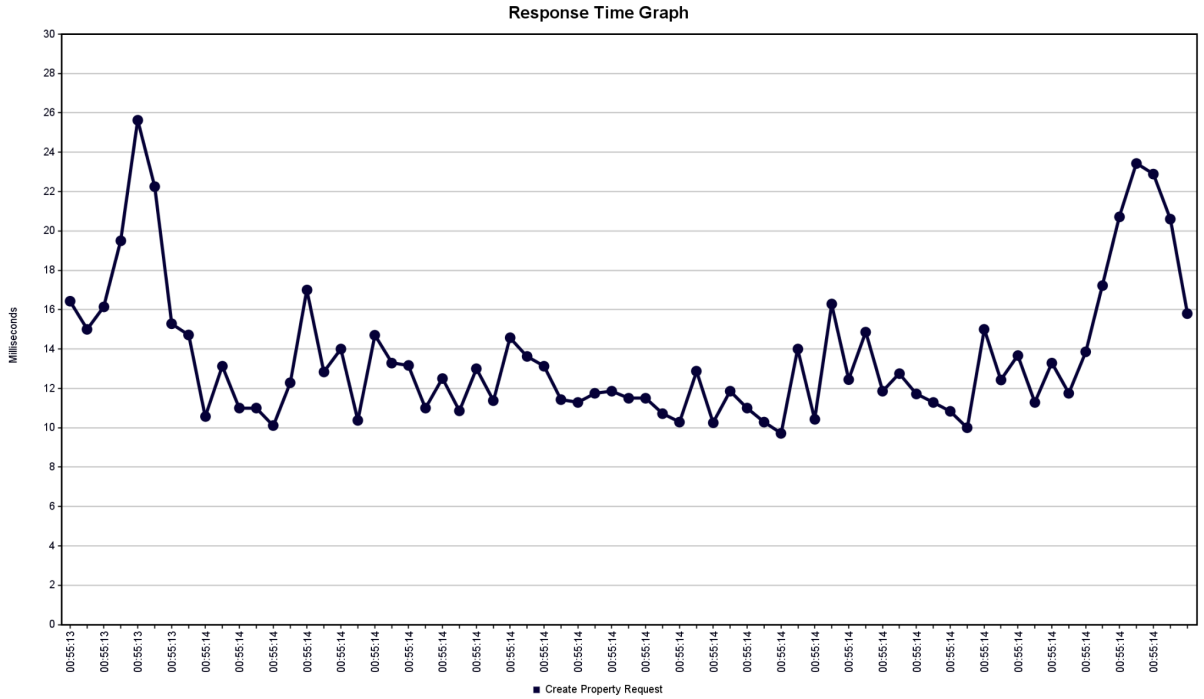
Broj thread-ova (user-a): 100



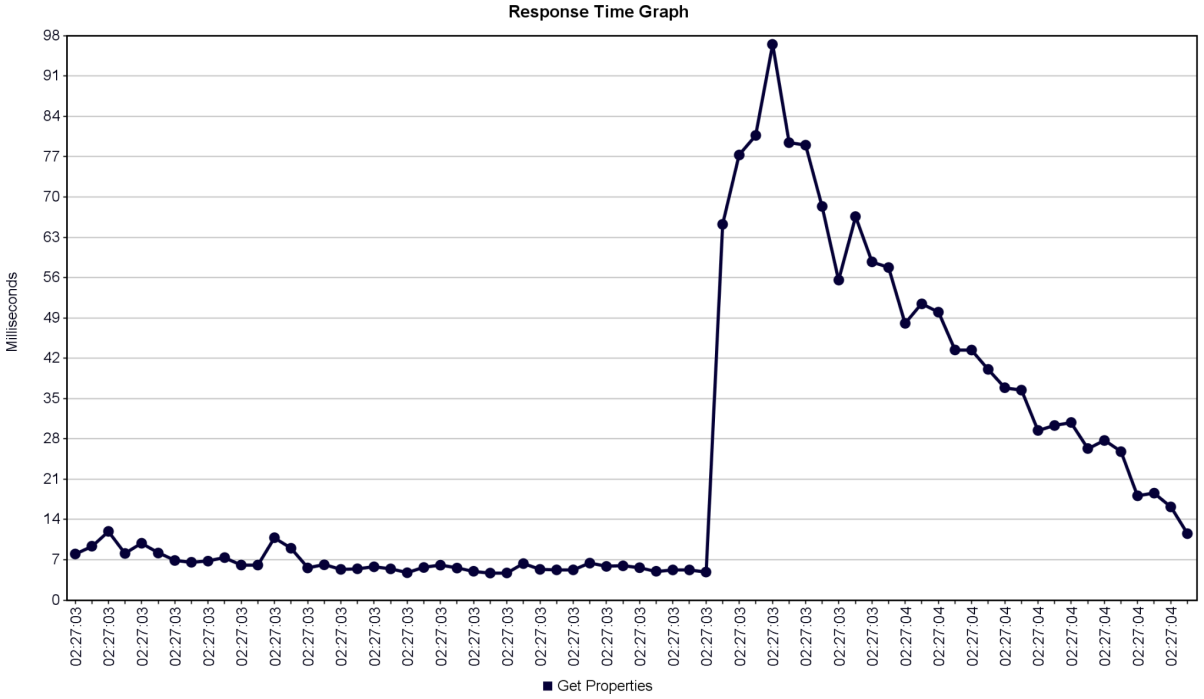

Broj thread-ova (user-a): 500



Broj thread-ova (user-a): 1000



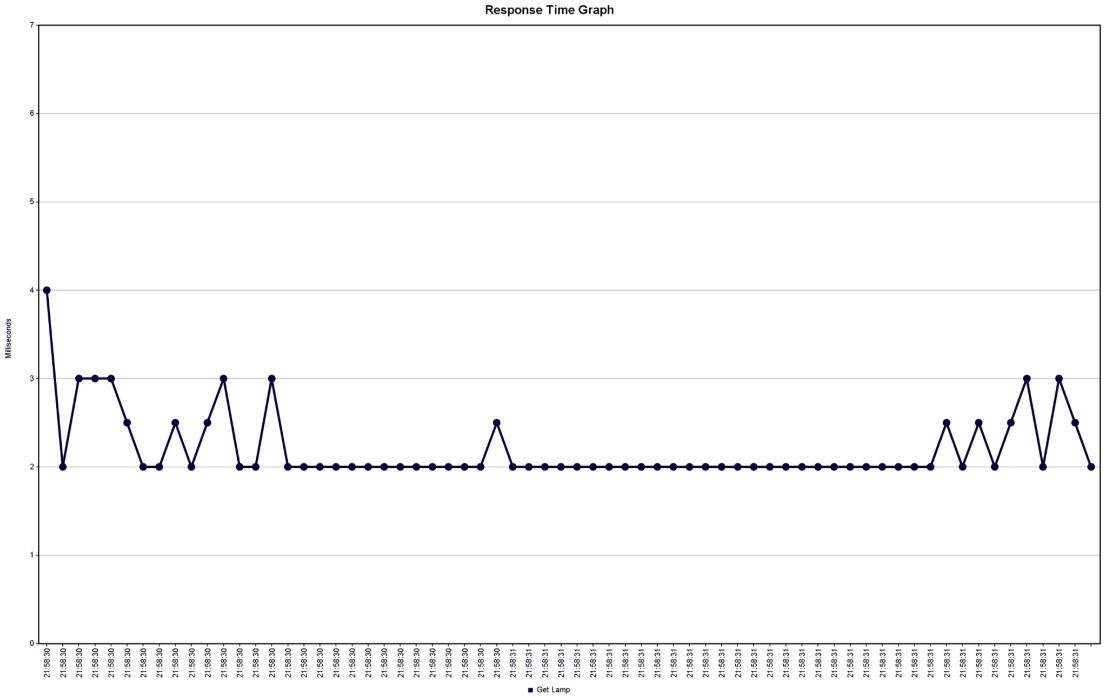

Broj thread-ova (user-a): 1300



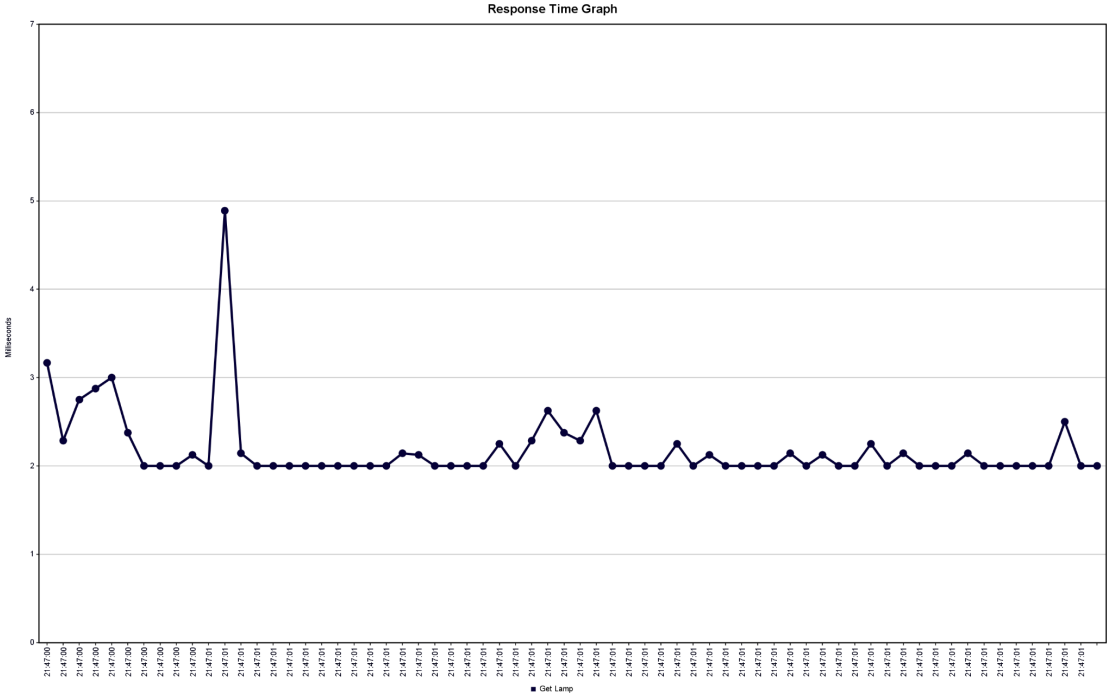
# Get Lamp

Prikaz vremena odziva sistema za funkcionalnost dobavljanja pametnog uredjaja “Lampa”. Sva testiranje za Get Lamp funkcionalnost su pokrenuta dok je istovremeno simuliran rad 50 pametnih uredjaja.

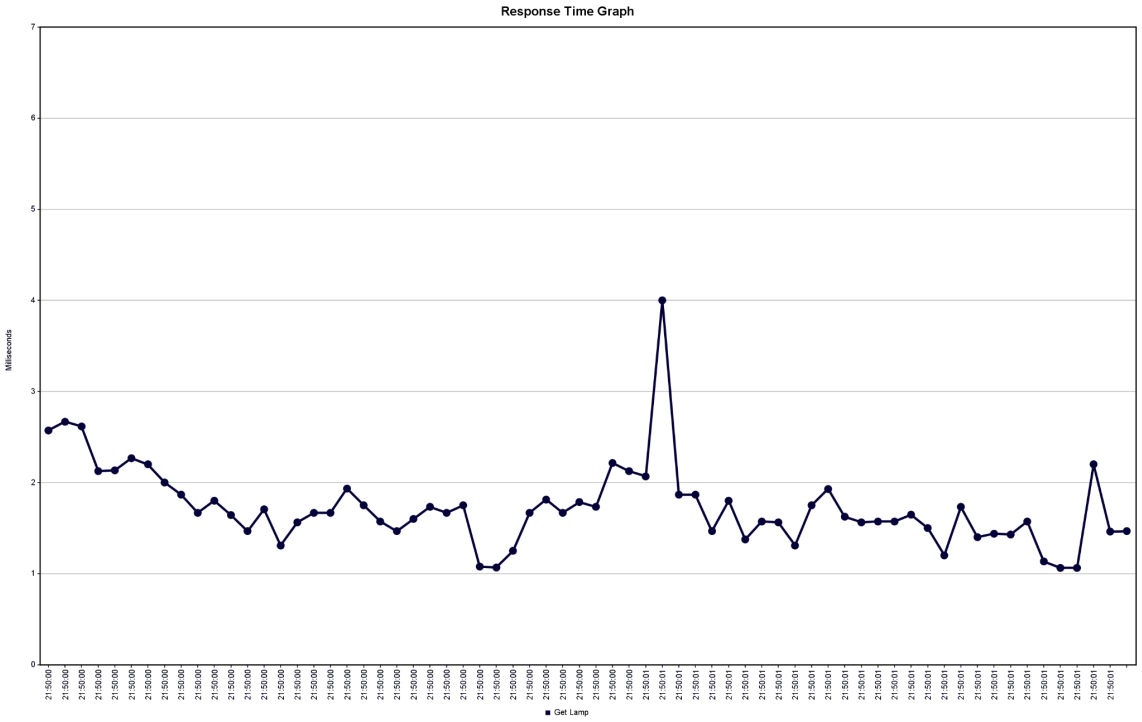
Broj thread-ova (user-a): 100



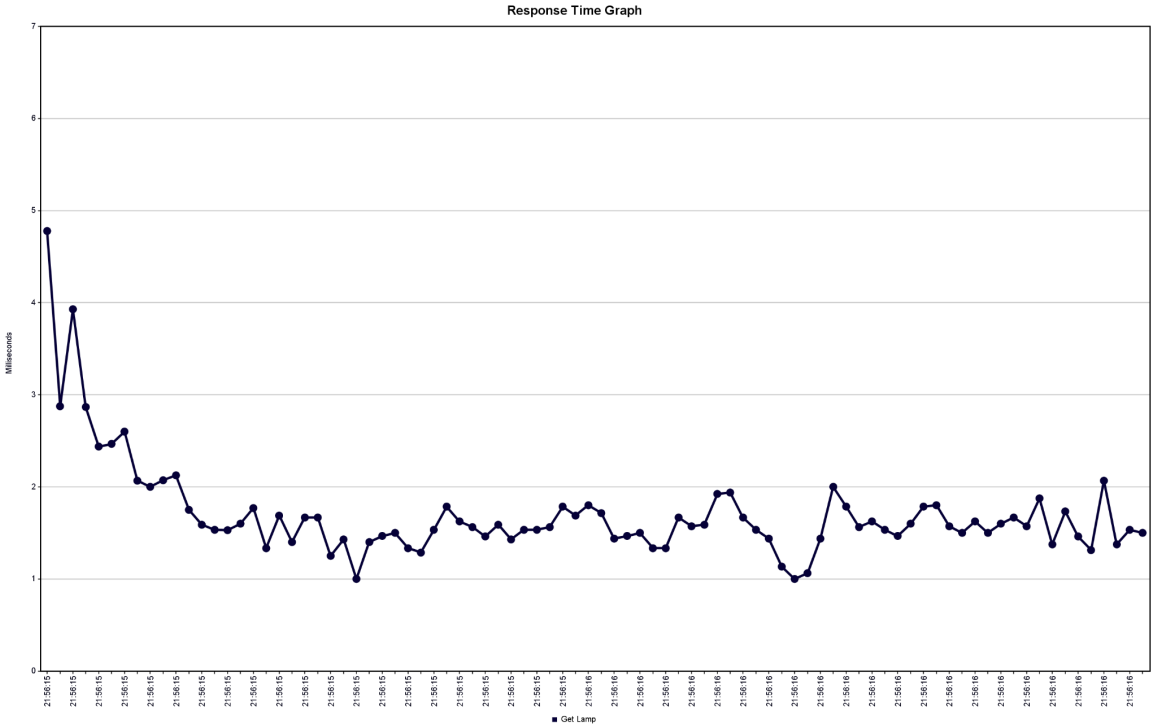

Broj thread-ova (user-a): 500




Broj thread-ova (user-a): 1000



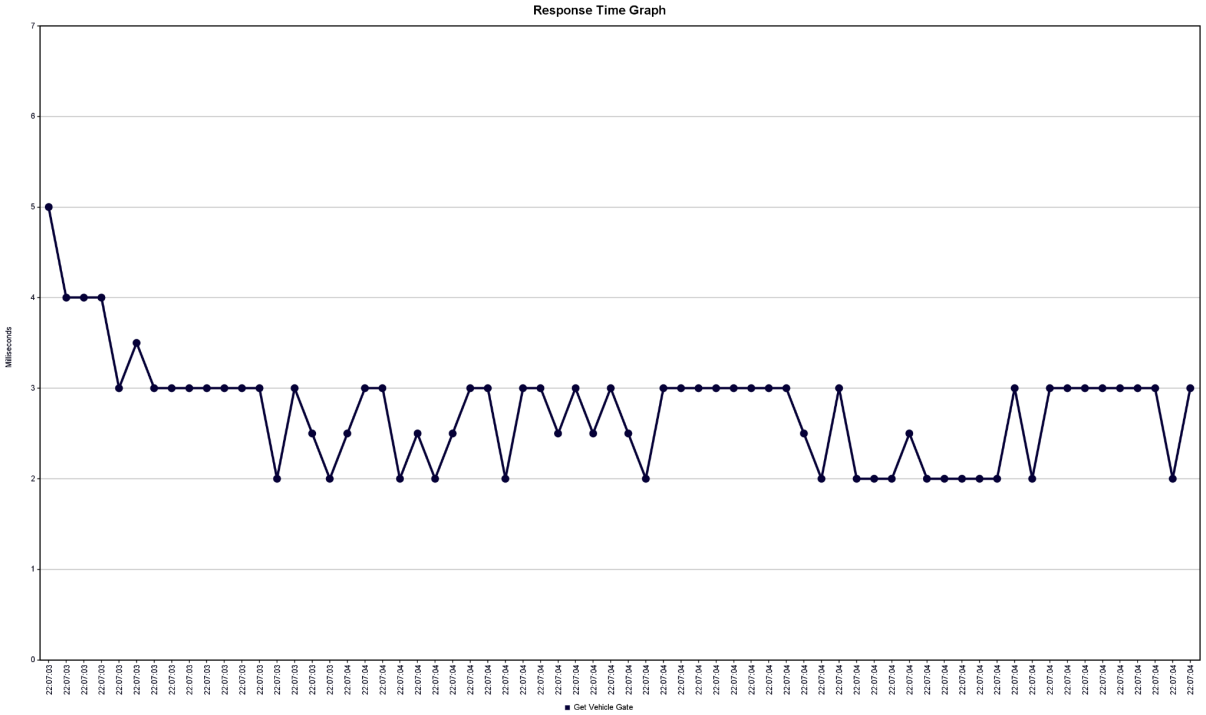

Broj thread-ova (user-a): 1300



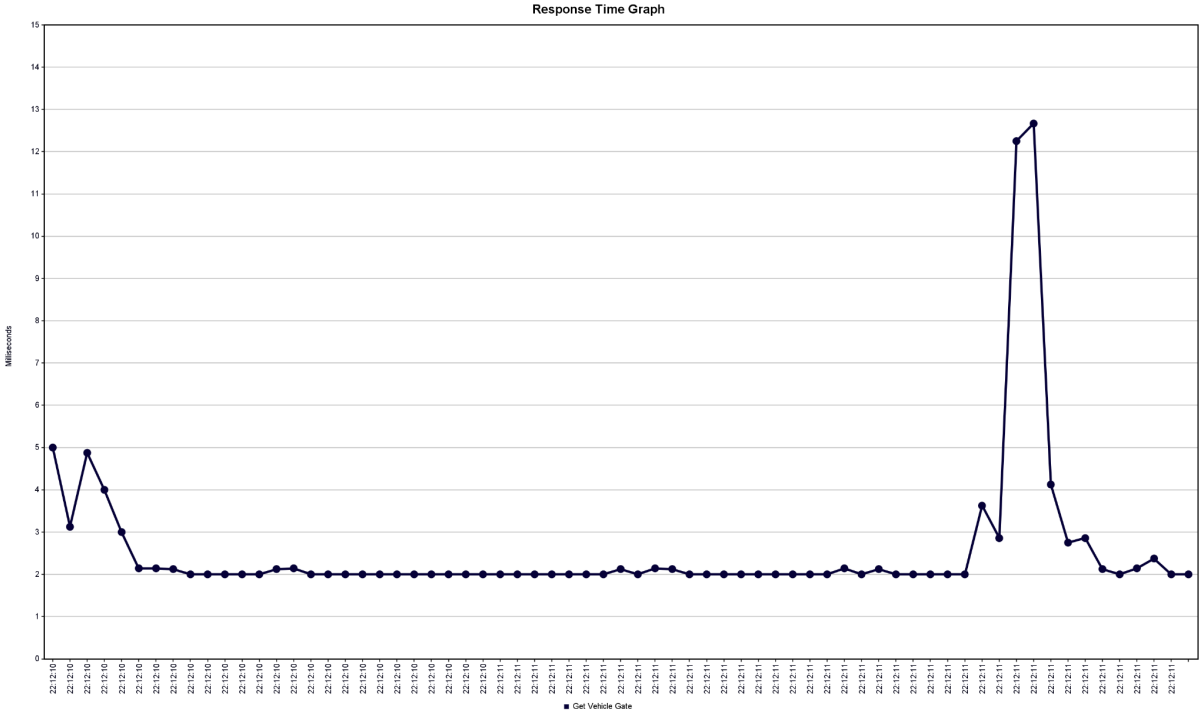
# Get Vehicle Gate

Prikaz vremena odziva sistema za funkcionalnost dobavljanja pametnog uredjaja “Kapija”. Sva testiranje za Get Vehicle Gate funkcionalnost su pokrenuta dok je istovremeno simuliran rad 50 pametnih uredjaja.

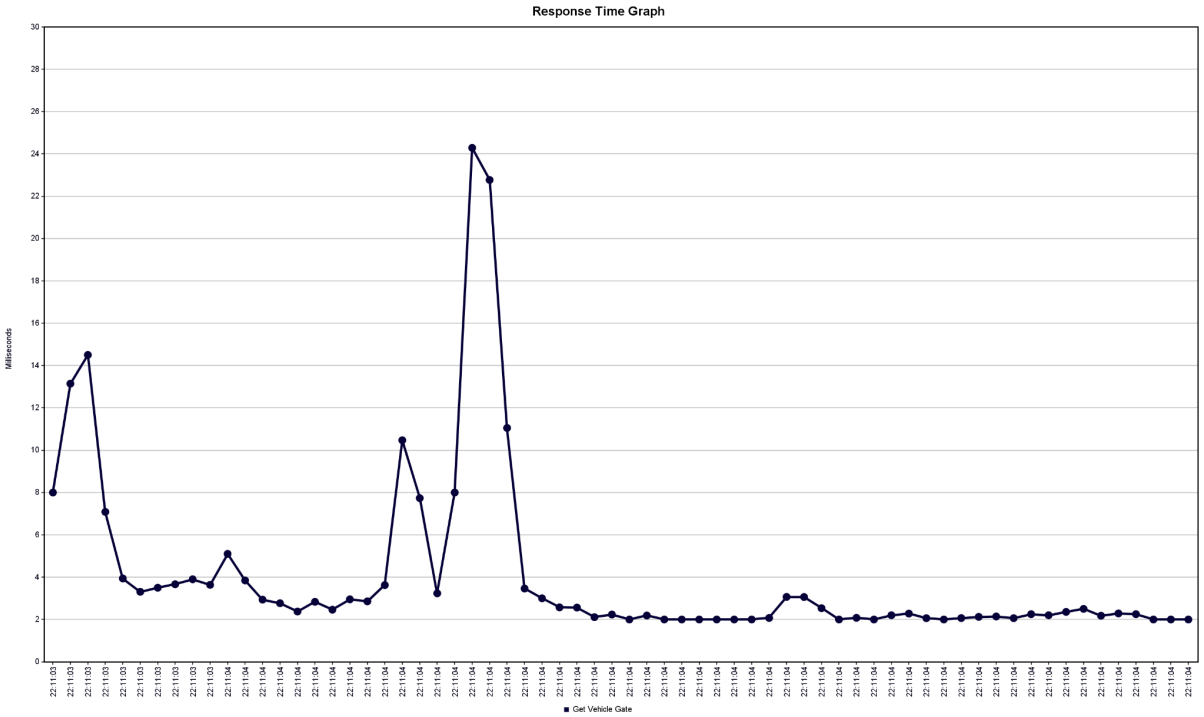
Broj thread-ova (user-a): 100



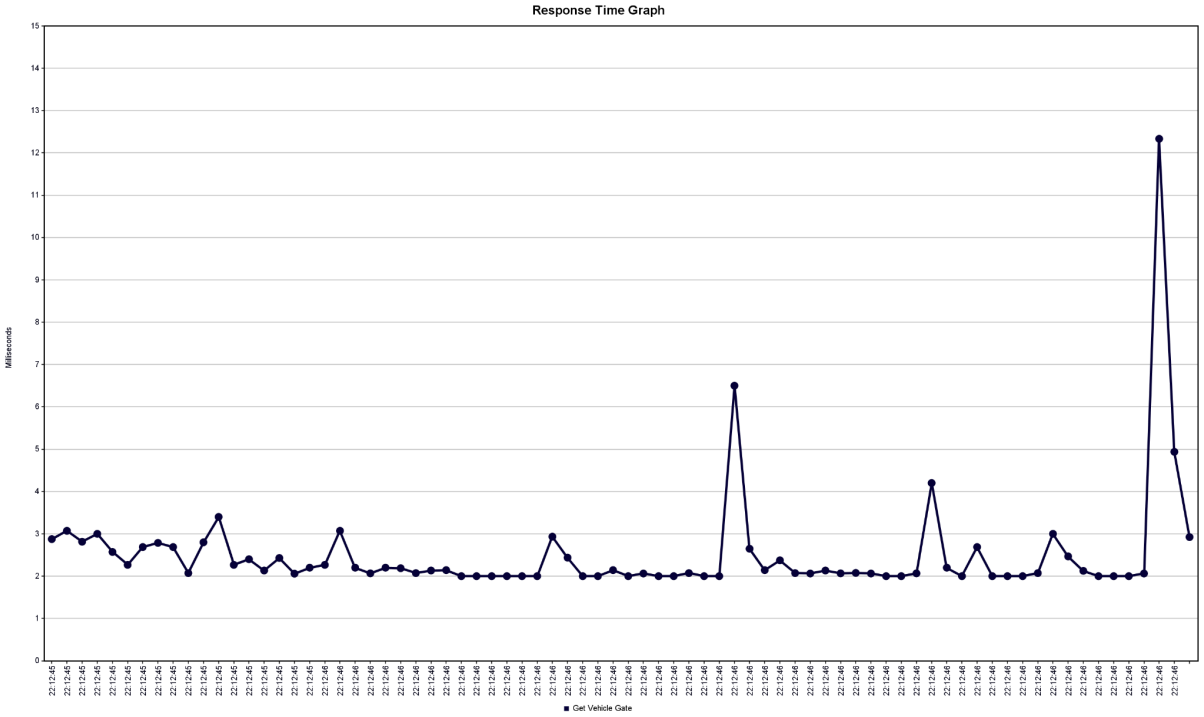

Broj thread-ova (user-a): 500



Broj thread-ova (user-a): 1000



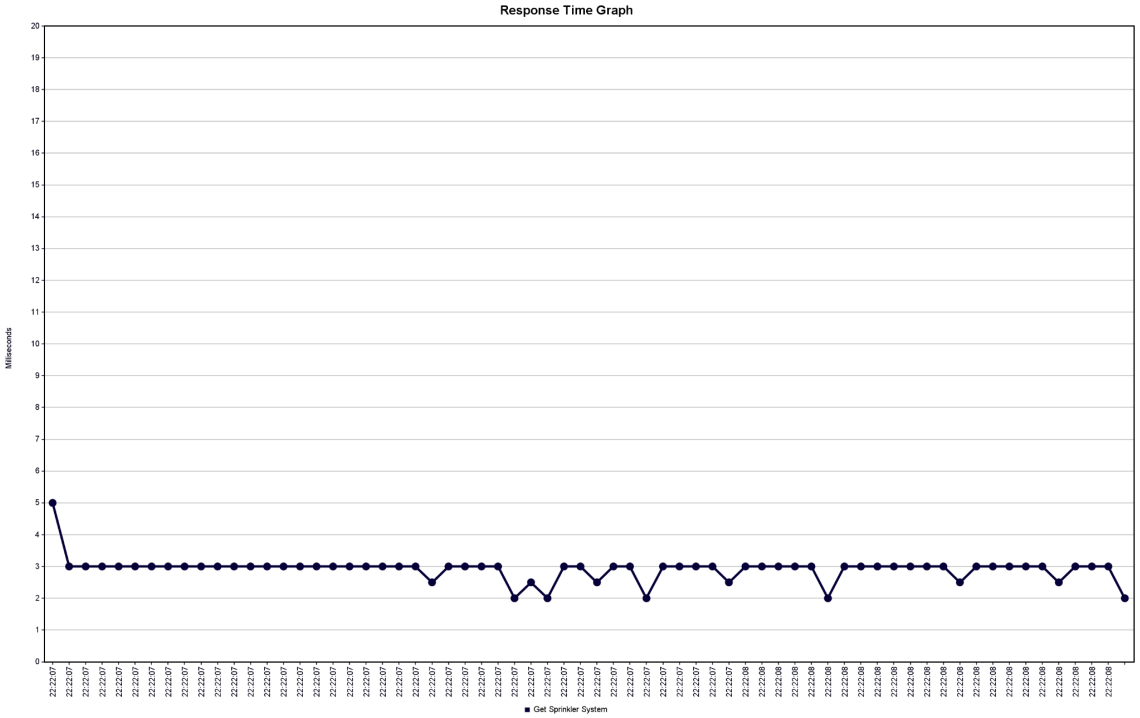

Broj thread-ova (user-a): 1300



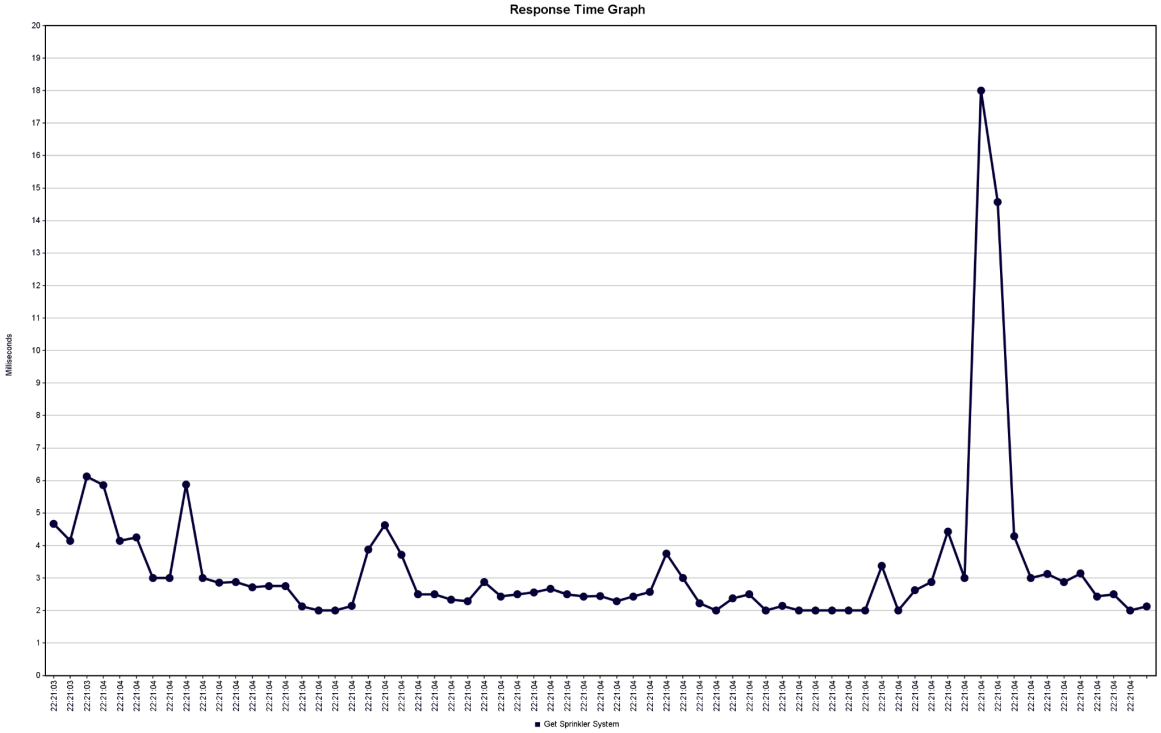
# Get Sprinkler System

Prikaz vremena odziva sistema za funkcionalnost dobavljanja pametnog uredjaja “Sistem prskalica”. Sva testiranje za Get Vehicle Sprinkler System funkcionalnost su pokrenuta dok je istovremeno simuliran rad 50 pametnih uredjaja.

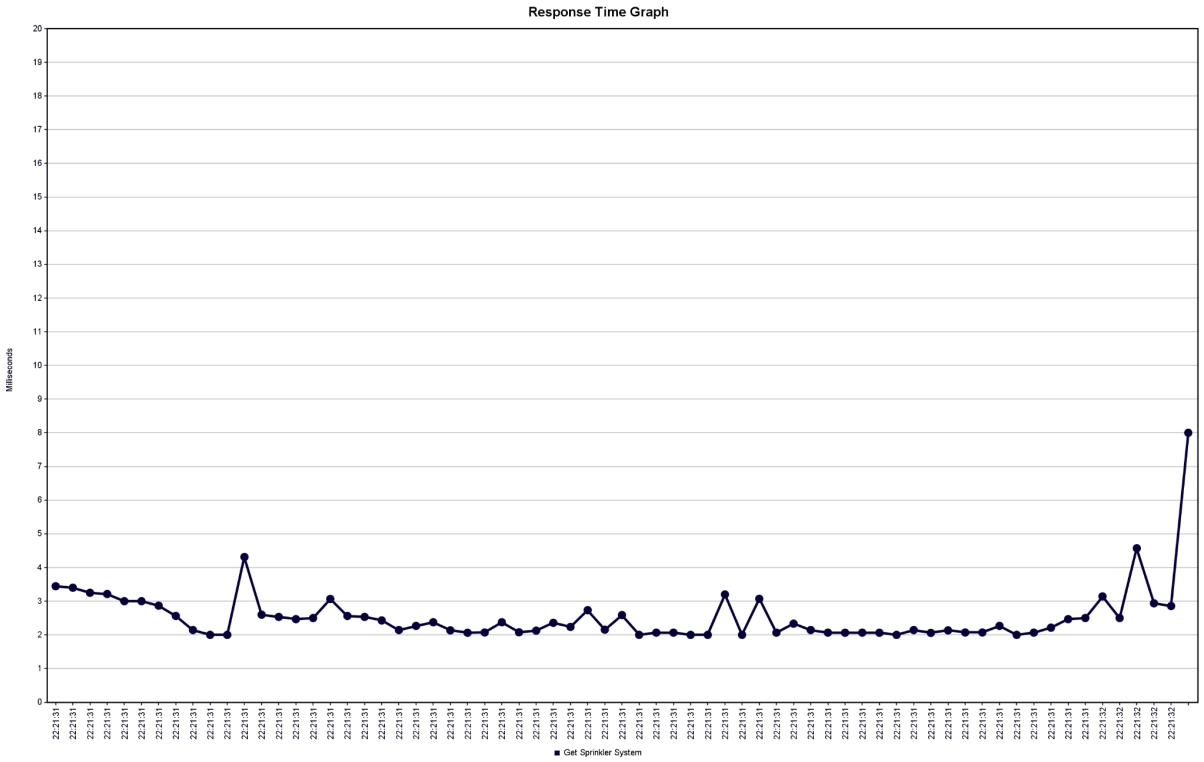
Broj thread-ova (user-a): 100



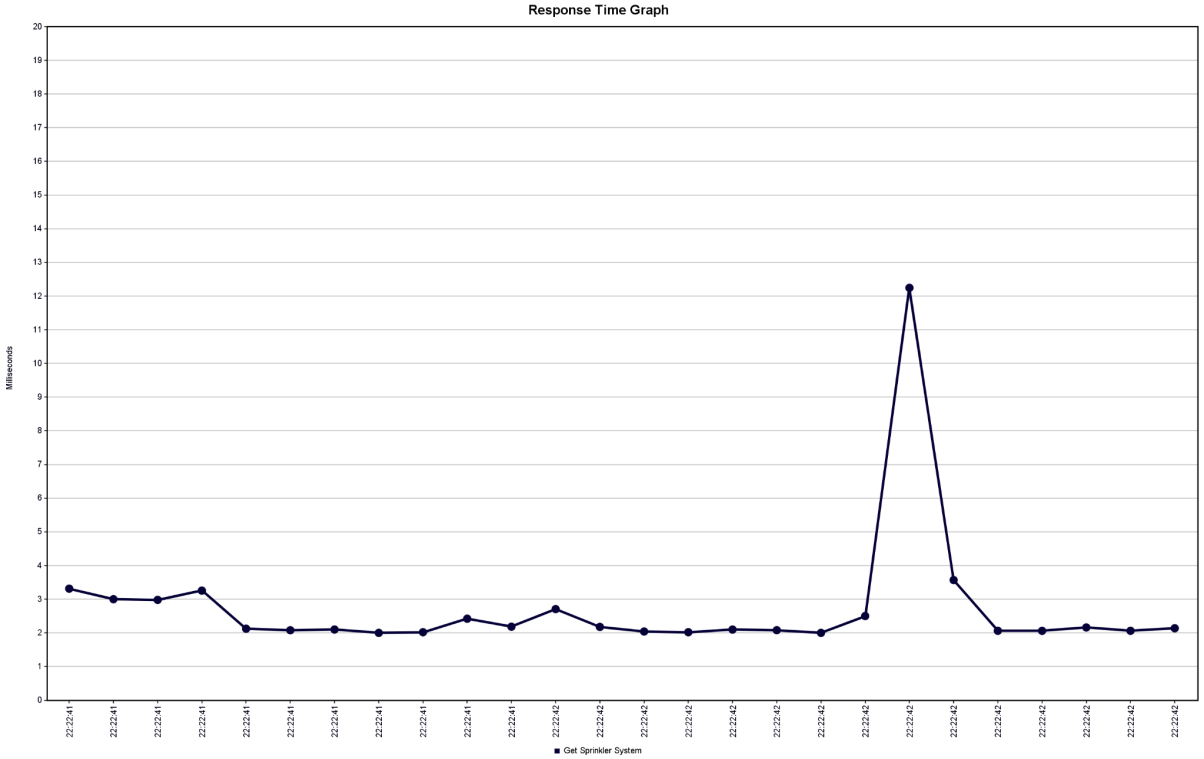

Broj thread-ova (user-a): 500



Broj thread-ova (user-a): 1000




Broj thread-ova (user-a): 1300



# Smart Device Performance Testing

Testiranje pokretanja simulacija za uredjaje. Tokom spokretanja simulacija, sistem je imao dobar odziv za 50 i 100 uredjaja, pa to nije prikazano u tabelama ispod. Testiranje je radjeno kroz iteracije (pokretano je 500, 1000 pa 10000 simulacija) i praceno je vrijeme koliko simulacijama treba da posalju informaciju nazad na front dio aplikacije.

Simulacije se pokrecu kao “runnables” i izvrsavaju se na svakih N sekundi (u zavisnosti od pametnog uredjaja). Ovde sa naisla na kompikaciju pamcenja stanja naseg pametnog uredjaja ( na primjer pamcenja registarskih tablica koje su trenutno u dvoristu). Da bih prevazisla ovaj problem, uvela sam postojanje klase State, za svaki pametni uredjaj. U ovoj klasi se cuvaju sve bitne informacij potrebne za smu simulaciju rada pametnog uredjaja.

Kada korisnik promeni neku akciju, u sustini se odigra promena tog State objekta, tako da simulacija (kada se sledeci put pokrene runnable) uzima podatke koje je pronasla u tom trenutku u stat funkciji. Na taj nacin, vreme odgovora na akciju korisnika je brzo.

Posto moramo da pamtimo za koje sve uredjaje smo vec pokrenili runnable, u pocetku to je pisano u listu. Medjutim, da bi se ubrzale performanse, uvedeno je koriscenje strukture HashMap. U ovoj strukturi se cuva id pametnog uredjaja i njegov state, tako da uz pomoc jedne strukture mogu da proverim da li postoji vec runnable za uredjaj i koji mu je state. Ako, kada dodje zahtev od korinika, ne postoji u HashMap-i kljuc sa id pametnog uredjaja, on se dodaje i pokrece runnable za taj uredjaj. Kada je runnable pokrenut, on proverava svoj state (preko id pametnog uredjaja).

Ovo je poboljsalo performanse, jer je vreme pretrage i pristupa kod strukture tipa lista O(n), dok je pristup kod HashMap-e O(1) jer se vrsi pristup preko kljuca unutar strukture. Posto je u pitanju programiranje sa vise niti, koriscena je konkurentna HashMapa, da ne bi doslo do stetnih preplitanja kada vise runnables zeli da ima pristup HashMap-i sa states objektima. Primjer ovakve HashMape:

```
ConcurrentHashMap<Long, LampState> states = new ConcurrentHashMap<>();
```

Takodje, kao jos jedno usko grlo sistema izdvojila bih ogranicenje Mqtt Brokera za broj publish koji moze da postigne u jednom trenutku. Takodje, bolja specifikacija racunara (vise RAM memorije i bolji procesor) doveli bi do vecih prformansi sistema i boljeg odziva.

U sledecim tabelama su prikazani rezultati kada pokrenemo veliki broj simulacija (500, 1000 i 10000), gdje svaka celija u tabeli predstavlja odstupanje vremena. Na primer, ako je uredjaj trebao da ocita novu vrednost za 10 sekundi, a to je ucinio za 12 sekundi, u korespodentnoj celiji ce biti upisana vrednost 2sek. Za akcije uredjaja, u svakoj korespodentnoj celiji tabele ce biti prikazano vreme za koje se funkcija odazvala (vreme kada je runnable promeni state, i vratio odgovor korisniku)

## Pametni uredjaj “Lampa”

| | 500 simulacija | 1000 simulacija | 5000 simulacija |
|---|---|---|---|
| Generisanje podataka (odstupanje od ocekivanog veremena) | 10ms | 200-400ms* | 1min-3min** |
| Bulb On/Off (vreme odziva na akciju) | 10ms | 500ms | 12s |
| Auto Mode On/Off (akcija) | 10ms | 500ms | 11-12s |

*u nekoliko navrata je kasnjenje bilo 800ms, ali takvo ponasanje nije konstantno (desilo se 1-2 puta)
**u nekoliko navrata je kasnjenje bilo 30 sekundi ili manje

## Pametni uredjaj “Kapija”

| | 500 simulacija | 1000 simulacija | 5000 simulacija |
|---|---|---|---|
| Generisanje podataka (odstupanje od ocekivanog veremena) | 9ms | 250-420ms | 1.5min-4min* |
| Turn Private mode On/Off (vreme odziva na akciju) | 10ms | 400ms | 11s |
| Open/Close (vreme odziva na akciju) | 9ms | 400ms | 12s |

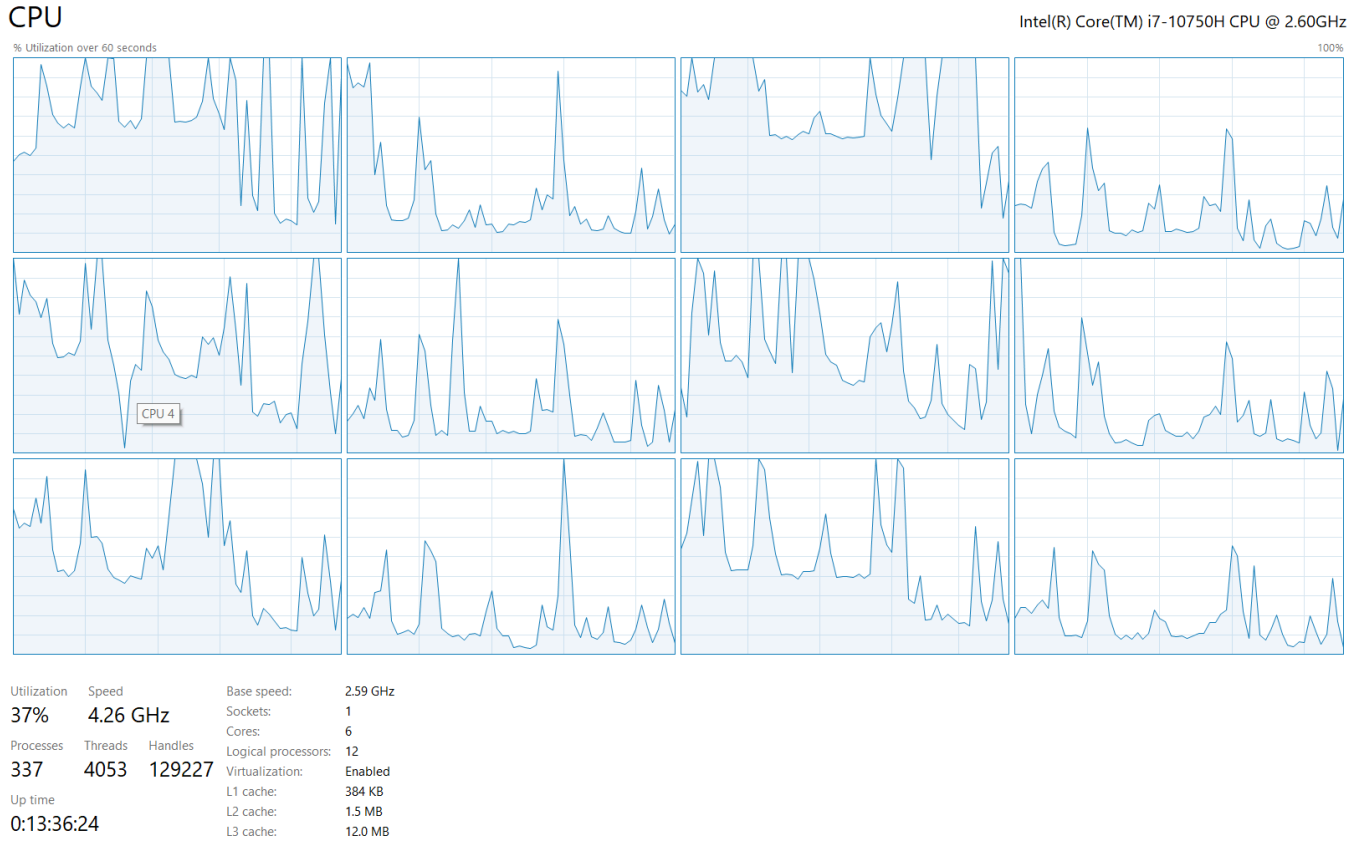
*u nekoliko navrata je kasnjenje bilo 40 sekundi ili manje

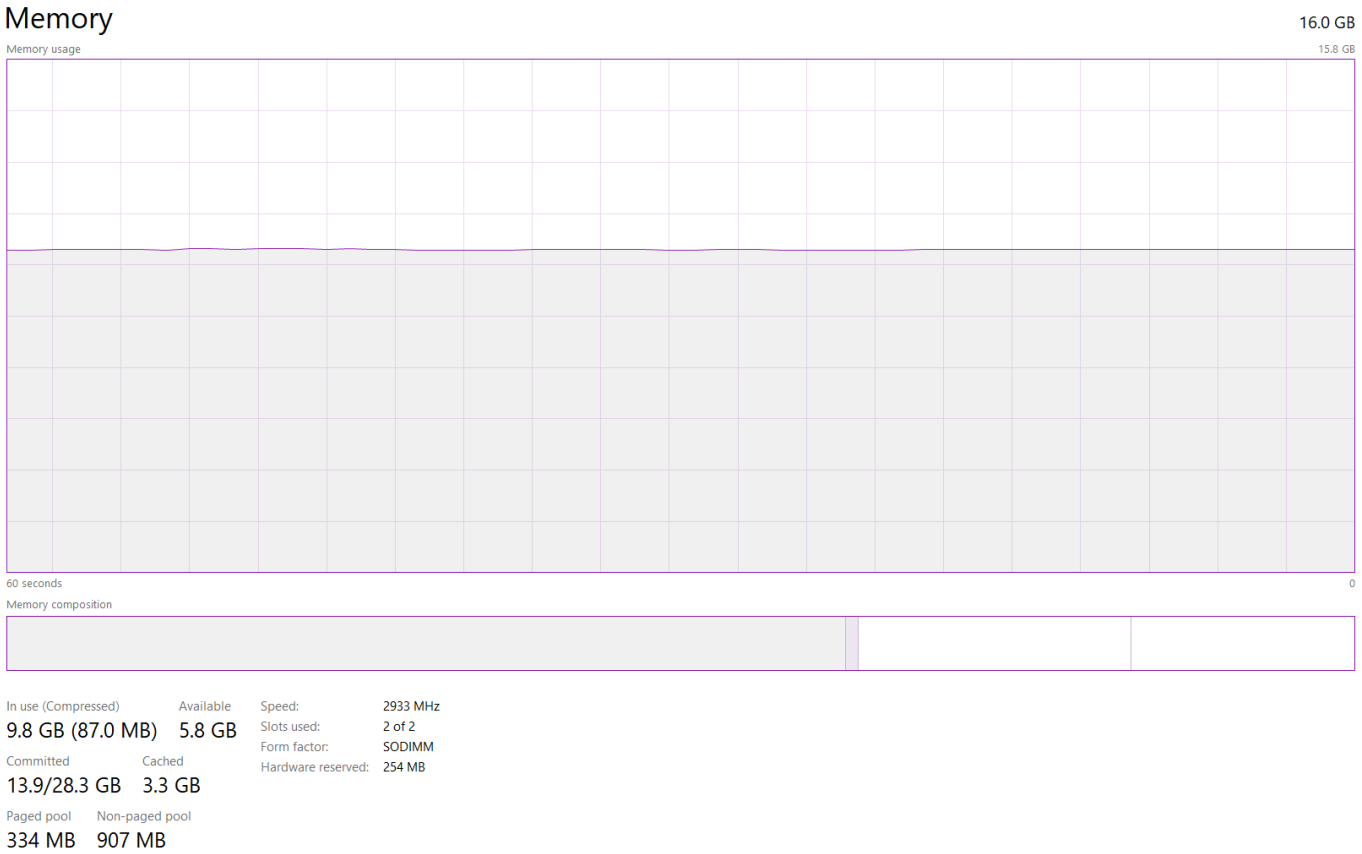
## Pametni uredjaj “Sistem Prskalica”

| | 500 simulacija | 1000 simulacija | 5000 simulacija |
|---|---|---|---|
| Generisanje podataka (odstupanje od ocekivanog veremena) | 5-10ms | 500ms | 2m-4m* |
| Turn On/Off (vreme odziva na akciju) | 10ms | 400ms | 10-12s |

*u nekoliko navrata je kasnjenje bilo 40 sekundi ili manje

Slika koja prikazuje opterecenje procesora kada je pokrenuto 5000 simulacija




Prikaz zauzeca memorije kada je pokrenuto 5000 simulacija