Usporedba performansi radnih okvira za izradu web usluga

Lončar, Ivan

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni diplomski studij procesnog računarstva

USPOREDBA PERFORMANSI RADNIH OKVIRA ZA IZRADU WEB USLUGA

Diplomski rad

Ivan Lončar

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 05.09.2023.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Ivan Lončar
Studij, smjer:	Diplomski sveučilišni studij Računarstvo, smjer Procesno računarstvo
Mat. br. Pristupnika, godina upisa:	D-509,
OIB studenta:	36615887833
Mentor:	prof. dr. sc. Marijan Herceg
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	izv. prof. dr. sc. Josip Job
Član Povjerenstva 1:	prof. dr. sc. Marijan Herceg
Član Povjerenstva 2:	izv. prof. dr. sc. Emmanuel-Karlo Nyarko
Naslov diplomskog rada:	Usporedba performansi radnih okvira za izradu web usluga
Znanstvena grana diplomskog rada:	**Telekomunikacije i informatika (zn. polje elektrotehnika)**
Zadatak diplomskog rada:	U radu je potrebno testirati različite radne okvire za izradu web usluga te usporediti odzivna vremena odgovora na upite, količinu obrađenih upita, postotak greške u odgovoru usluga, veličinu docker spremnika u kojima će se aplikacija podignuti. U radu je potrebno koristiti statički tipiziran, objektno orijentirani jezik Java, statički tipiziran, proceduralni jezik Golang i dinamički tipiziran, objektno orijentirani jezik Python. Pri kreiranju web usluga za Javu potrebno je koristiti Spring Boot i Quarkus framework, za Golang Gin and Gonic i Echo, a za Python Flask i FastApi.
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	05.09.2023.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 14.09.2023.

Ime i prezime studenta:	Ivan Lončar
Studij:	Diplomski sveučilišni studij Računarstvo, smjer Procesno računarstvo
Mat. br. studenta, godina upisa:	D-509,
Turnitin podudaranje [%]:	1

Ovom izjavom izjavljujem da je rad pod nazivom**: Usporedba performansi radnih okvira za izradu web usluga**

izrađen pod vodstvom mentora prof. dr. sc. Marijan Herceg

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Web servisi su tehnologija koja omogućava komunikaciju i razmjenu podataka između različitih softverskih aplikacija putem interneta. Oni pružaju standardizirani način za interakciju i dijeljenje informacija među softverskim sustavima, neovisno o njihovoj arhitekturi, platformi ili programskom jeziku [1]. Web servisi se temelje na skupu otvorenih standarda i protokola poput XML-a (engl. eXtensible Markup Language), SOAP-a (engl. Simple Object Access Protocol) i REST-a (engl. Representational State Transfer) [2]. Ti standardi definiraju strukturu, prijenos i obradu podataka tijekom komunikacije između aplikacija.

Web servisi se mogu podijeliti u dvije glavne vrste [3]:

- Temeljeni na SOAP-u koji omogućuje strukturiranu i standardiziranu komunikaciju između aplikacija putem različitih mrežnih protokola poput HTTP-a (engl. Hypertext Transfer Protocol), SMTP-a (engl. Simple Mail Transfer Protocol) i drugih. Web servisi temeljeni na SOAP-u koriste XML za prikazivanje podataka i definiraju operacije i formate poruka putem WSDL-a (engl. Web Services Description Language). Oni slijede formalniji i stroži pristup u razmjeni podataka.
- Temeljeni na REST-u koji koristi postojeći HTTP protokol za komunikaciju i HTTP metode poput GET, POST, PUT i DELETE za izvođenje operacija nad resursima identificiranim jedinstvenim URL-om (engl. Uniform Resource Locator). Najčešće koriste formate poput JSON-a (engl. JavaScript Object Notation) ili XML-a.

Web servisi se široko koriste u različitim scenarijima, uključujući [4]:

- Integracija aplikacija u kojoj web servisi omogućavaju komunikaciju i razmjenu podataka između različitih softverskih aplikacija, olakšavajući integraciju i sinkronizaciju podataka između sustava.
- SOA (engl. Service-Oriented Architecture) u kojoj su web usluge ključna komponenta i prema kojoj se aplikacije dizajniraju kao modularne i ponovno iskoristive usluge koje se mogu kombinirati za izgradnju većih sustava.
- Pružaju API-je (engl. Application Programming Interfaces) koji omogućuju mobilnim i web aplikacijama pristup i korištenje podataka s udaljenih poslužitelja ili usluga.
- B2B (engl. Business-to-Business) gdje web usluge omogućuju besprijekornu komunikaciju i razmjenu podataka između organizacija, olakšavajući B2B interakcije i transakcije.

Web usluge omogućuju interoperabilnost, integraciju i suradnju između različitih softverskih sustava, potičući učinkovitu i standardiziranu komunikaciju putem interneta. Ovisno o problemskoj domeni, vrsti analize, obrade i izlaganja podataka, programeri se mogu odlučiti razvijati web usluge u jednom od dostupnih programskih jezika te po potrebi koristiti radni okvir tog programskog jezika.

Radni okvir je skup alata, biblioteka, pravila i konvencija koji omogućuju razvoj softvera olakšavajući programerima izgradnju aplikacija. Radni okvir pruža strukturu i temeljne funkcionalnosti koje omogućuju programerima da se usredotoče na specifične zadatke umjesto da ponovno izmišljaju osnovne komponente [5].

Glavne značajke radnog okvira su [6]:

- Ima svoju strukturu i konvencije za organizaciju koda. To olakšava čitljivost i održavanje aplikacija te omogućava konzistentnost među različitim projektima.
- Daje već izgrađene komponente i module koji se mogu ponovno koristiti u različitim dijelovima aplikacije ili čak između različitih projekata. Ovo pomaže u smanjenju vremena razvoja i poboljšanju produktivnosti.
- Nudi mehanizme za upravljanje ovisnostima između različitih komponenti i biblioteka. Ovo olakšava integraciju s vanjskim resursima i ubrzava proces razvoja.
- Daje sigurnosne mehanizme i smjernice za zaštitu aplikacija od poznatih ranjivosti. Ovo pomaže programerima da izbjegnu uobičajene sigurnosne propuste.
- Dolazi s razvojnim alatima, poput alata za ispitivanje, profiliranje i automatsko testiranje, koji pomažu u razvoju i održavanju aplikacija.

Korištenje radnog okvira olakšava razvoj, povećava produktivnost i omogućava programerima da se fokusiraju na rješavanje poslovnih problema umjesto na temeljne tehničke detalje.

U ovom radu testirane su i uspoređene performanse web usluga razvijenih u šest različitih radnih okvira. Korištena su tri programska jezika; *Java*, *Golang* i *Python* i dva radna okvira za svaki, za *Javu Spring Boot* i *Quarkus*, za *Golang Gin* i *Echo*, te za *Python FastAPI* i *Flask*. Testiranje performansi web usluga je proces provjere performansi i skalabilnosti web usluga kako bi se utvrdilo kako se usluge ponašaju pod određenim opterećenjem i uvjetima korištenja. Cilj testiranja performansi je identificirati moguće slabosti ili probleme u performansama web usluga i osigurati da usluga može podržati očekivani broj korisnika i zahtjeva [7].

Glavni aspekti testiranja performansi web usluga uključuju [8]:

- Testiranje opterećenja koje uključuje provjeru kako se web usluga ponaša pod različitim razinama opterećenja. Ovo testiranje uključuje reakcije usluge na veliki broj istovremenih korisnika, veliki broj zahtjeva ili veliku količinu podataka.
- Testiranje skalabilnosti kojim se provjerava kako se web usluga ponaša kada se promijeni broj korisnika ili opterećenje. Ovim testiranjem cilj je utvrditi je li usluga u stanju rasti i prilagoditi se povećanju zahtjeva.
- Vrijeme odziva kojim se mjeri vrijeme koje web usluga treba za obradu zahtjeva i generiranje odgovora. Ovim testiranjem cilj je osigurati da usluga pruža brz i optimalan odziv korisnicima.
- Skaliranje infrastrukture uključuje testiranje kako će se infrastruktura koja podržava web uslugu skalirati, poput servera, baze podataka i mrežne infrastrukture. Ovim testiranjem cilj je osigurati da infrastruktura može podržati očekivano opterećenje i zahtjeve.
- Stres testiranje provjerava granice izdržljivosti usluge što uključuje testiranje ekstremnih uvjeta, poput vrlo visokog opterećenja ili dugotrajnog izvršavanja zahtjeva.

Testiranje performansi web usluga može se provoditi koristeći različite alate i tehnike, kao što su alati za opterećenje i testiranje performansi, simuliranje stvarnog prometa korisnika ili automatsko generiranje velikog broja zahtjeva [9]. U ovom radu je korišten *Apache JMeter*. Kroz testiranje performansi, mogu se identificirati problemi poput sporih odgovora, preopterećenosti sustava, problema sa skalabilnošću ili slabih performansi pri visokom opterećenju. Rezultati testiranja omogućuju timu za razvoj i operacije da optimiziraju web uslugu i osiguraju njenu pouzdanu i učinkovitu izvedbu.

Cilj rada je prikazati numeričke vrijednosti analize vremena obrade podataka i količine upita u različitim programskim jezicima i radnim okvirima. U ovisnosti o domeni problema koji pojedina web usluga obrađuje, i na koju utječe puno faktora; opterećenost - očekivani broj korisnika aplikacije, brzina razvijanja aplikacije u ovisnosti o zrelosti programskog jezika i radnog okvira, dostupnost resursa za održavanje koda u odabranom programskom jeziku i radnom okviru kao i vrijeme obrade specifičnog problema u određenom programskom jeziku, programer se može odlučiti za programski jezik i radni okvir koji najbolje odgovara potrebama njegove aplikacije. Numeričke vrijednosti i analiza dostupne u radu mogu pomoći u odabiru programskog jezika i radnog okvira prije nego se aplikacija počne razvijati. U drugom poglavlju opisani su postojeći radovi koji se bave testiranjem performansi različitih radnih okvira. U trećem poglavlju dan je opis rješenja za usporedbu radnih okvira dok su u četvrtom poglavlju prikazani rezultati i analiza testiranja. U petom poglavlju su predstavljeni zaključci rada.

2. PREGLED INTERNET STRANICA ZA USPOREDBU RADNIH OKVIRA

Na internetu postoji nekoliko web stranica i platformi koje pružaju usporedbe performansi različitih radnih okvira. Neke od popularnih web stranica na kojima možete pronaći takve usporedbe su:

- *TechEmpower Framework Benchmarks* [10] je poznati projekt koji uspoređuje performanse različitih web radnih okvira. Pruža rezultate testiranja za razne scenarije, uključujući opterećenje baze podataka, obradu JSON-a, server-side rendering itd.
- *Open-Source Frameworks Performance* je GitHub projekt [11] koji sadrži usporedbe performansi različitih otvorenih radnih okvira. Pruža testove performansi i rezultate za razne jezike i okvire.
- *StackShare* web stranica [12] nije specifično usmjerena na usporedbu performansi, ali pruža informacije o različitim radnim okvirima, uključujući ocjene korisnika, popularnost i tehnologije koje koriste.

Neki od testnih scenarija koje koriste na navedenim stranicama uključuju:

- *Plaintext* je test koji mjeri brzinu odgovora web servera kada se šalje zahtjev za prikazivanje jednostavnog odgovora. Time se daje uvid u brzinu obrade osnovnog HTTP zahtjeva.
- *JSON Serijalizacija* je test koji mjeri brzinu serijalizacije (pretvorbe u JSON format) objekta na serveru i slanja kao odgovora. Time se daje uvid u testiranje performansi radnih okvira u obradi JSON podataka.
- *Pojedinačni upit na bazu podataka* je test kojim se mjeri brzina izvršavanja pojedinačnog upita na bazu podataka. Time se daje uvid u informacije o performansama radnih okvira u interakciji s bazom podataka.
- *Višestruki upiti na bazu podataka* je test koji mjeri brzinu izvršavanja više upita na bazu podataka kako bi se simulirala složenija operacija s više podataka. Time se daje uvid u testiranje skalabilnosti i brzinu izvršavanja radnih okvira u složenijim scenarijima baze podataka.
- *Fortunes* je scenarij u kojem se testira brzina generiranja HTML stranice koja prikazuje popis "sretan" (engl. fortune) poruka iz baze podataka. Navedeni scenarij pruža uvid u performanse radnih okvira u generiranju HTML sadržaja.

- *Usmjeravanje* je scenarij kojim se provjerava brzina usmjeravanja zahtjeva u radnom okviru. Testira se kako radni okvir usmjerava dolazne zahtjeve prema odgovarajućim upravljačima ili rutama.
- *Prikaz predloška* je test kojim se provjerava brzina generiranja dinamičkog sadržaja predlošcima. U ovom scenariju se mjeri performansa radnog okvira u generiranju HTML ili drugih formata temeljenih na predlošcima.
- *Predmemoriranje* je testni scenarij kojim se procjenjuju performanse radnog okvira u vezi s mehanizmima predmemoriranja. U ovom scenariju se mjeri se brzina pristupa i vraćanja podataka iz predmemorije.
- *Istodobnost* je scenarij kojim se testiraju performanse radnog okvira pri obradi višestrukih zahtjeva istodobno. U ovom scenariju se mjeri se vremensko trajanje i resursi potrebni za rukovanje s više konkurentnih zahtjeva.
- *Posluživanje statičke datoteke* je scenarij koji provjerava brzinu servisiranja statičkih datoteka kao što su slike, CSS (engl. Cascading Style Sheets) i *JavaScript* datoteke. U ovom scenariju se mjeri se brzina prenošenja i dostupnost statičkog sadržaja putem radnog okvira.

Važno je napomenuti da usporedbe performanse mogu biti subjektivne i ovise o različitim faktorima poput vrste aplikacije, hardverske konfiguracije, konkretnog koda i postavki. Stoga je potrebno razmotriti više izvora i provjeriti specifične testove i scenarije koji su vam relevantni prije donošenja zaključaka o performansama radnih okvira.

3. OPIS RJEŠENJA ZA USPOREDBU RADNIH OKVIRA

U ovom diplomskom radu napravljena je usporedba radnih okvira programskih jezika *Java*, *Golang* i *Python*. Korišteni radni okviri programskog jezika *Java* su *Spring Boot* i *Quarkus*; *Golang*-a *Gin* i i *Echo*, *Python*-a *FastAPI* i *Flask*.

3.1. Java

Java je jezik visoke razine, objektno orijentiran, statički tipiziran i kompiliran jezik koji je razvijen s ciljem jednostavnosti, prenosivosti i sigurnosti. *Java* je osmišljena kako bi omogućila pisanje aplikacija jednom i izvođenje bilo gdje, što znači da se programi napisani u *Javi* mogu pokretati na različitim platformama bez potrebe za prevođenjem [13]. *Java* je izgrađena na konceptu objekata, što znači da se sve u Javi smatra objektom. Korištenje objekata omogućuje modularnost, ponovnu upotrebu koda i olakšava razvoj i održavanje aplikacija. *Java* je također statički tipiziran jezik, što znači da se provjerava ispravnost tipova podataka tijekom kompilacije. Jedna od ključnih karakteristika *Jave* je *Java* virtualna mašina (engl. *Java* Virtual Machine - JVM), koja omogućuje izvršavanje *Java* programa na različitim platformama. Kada se *Java* program prevede u *bytecode*, koji je međukod JVM-a, može se izvoditi na bilo kojem uređaju koji ima instaliranu JVM. *Java* je također poznata po svojoj velikoj standardnoj biblioteci, koja sadrži velik broj klasa i metoda koje olakšavaju razvoj aplikacija. Ova biblioteka pruža funkcionalnosti poput obrade nizova, rad s mrežama, rad s bazama podataka, grafičkog korisničkog sučelja i još mnogo toga. *Java* se široko koristi za razvoj različitih vrsta aplikacija, uključujući desktop aplikacije, web aplikacije, mobilne aplikacije, velike poslovne sustave i igre. Njena popularnost, robusnost, sigurnost i prenosivost čine je jednim od najpopularnijih programskih jezika u industriji softvera. Korištena verzija *Jave* za lokalni razvoj u ovom radu je openjdk 19.0.2 2023-01-17 i *JRE OpenJDK Runtime Environment Temurin-19.0.2+7 (build 19.0.2+7)*. Korištena verzija *Java Docker* slike je *openjdk:19*.

3.1.1. Spring Boot

Spring Boot je projekt otvorenog programskog koda i radni okvir za razvoj *Java* aplikacija. On pruža brz i jednostavan način za stvaranje samostalnih, proizvodno spremnih aplikacija. *Spring Boot* temelji se na *Spring* radnom okviru i omogućava programerima da se usredotoče na razvoj poslovne logike umjesto na složenost konfiguracije [14].

Neke ključne karakteristika *Spring Boot*-a:

- Konvencija iznad konfiguracije kojim su mnogi konfiguracijski detalji automatski riješeni uz pretpostavke i konvencije koje omogućuju lakši i brži početak razvoja i umanjuje potrebu za ručnom konfiguracijom.
- Samostalnost koja omogućuje izgradnju samostalnih aplikacija koje sadrže ugrađeni web poslužitelj i sve potrebne ovisnosti, pa aplikacija može biti pokrenuta izravno kao jedna samostalna JAR (engl. Java ARchive) datoteka bez potrebe za vanjskim web poslužiteljem.
- Web okvir dolazi s mnogim ugrađenim značajkama i modulima koji olakšavaju razvoj aplikacija. Primjerice, podržava ugrađene baze podataka, upravljanje konfiguracijom, sigurnost, testiranje i još mnogo toga.
- Arhitektura mikro usluga, gdje svaka mikro usluga može biti samostalna aplikacija.
- Injekcija ovisnosti omogućuje odvajanje funkcionalnosti klasa [15] .
- Inverzija kontrole je načelo dizajna koje omogućava labavo povezivanje klasa čime ih čini lakim za održavanje i testiranje. Odnosi se na prijenos kontrole nad objektima i njihovim ovisnostima iz glavnog programa u spremnik ili okvir [16].

Spring Boot je postao vrlo popularan među programerima *Java* aplikacija zbog svoje jednostavnosti, brzine razvoja, efikasnosti i podrške za moderne razvojne prakse. Korištena verzija *Spring Boot* web okvira u ovom radu je *3.0.4*.

3.1.2. Quarkus

Quarkus je projekt otvorenog programskog koda i radni okvir za razvoj *Java* aplikacija, posebno usmjeren na izgradnju brzih, laganih i efikasnih mikro usluga i aplikacija u računalnom oblaku. *Quarkus* je dizajniran da omogući visoku produktivnost razvoja uz minimalnu potrošnju resursa i brzo pokretanje same aplikacije.

Neke od ključnih karakteristika *Quarkusa* [17]:

- Brzo pokretanje jer *Quarkus* koristi tehnike kao što su *GraalVM* i pred kompilacija da bi postigao iznimno brzo vrijeme pokretanja. Time se omogućuje brzo pokretanje aplikacija i optimizirano iskorištavanje resursa.
- Razvoj mikro usluga i aplikacija u računalnom oblaku jer pruža integracijske alate za izgradnju, konfiguraciju i upravljanje mikro uslugama te podržava karakteristike kao što su pokretanje aplikacija u *Docker* kontejnerima, upravljanje konfiguracijom, skalabilnost i otpornost na greške.

- Podržava reaktivno programiranje što omogućuje razvoj aplikacija koje su odzivne, skalabilne i otporne na visoke opterećenja. Koristi tehnologije poput *Reactive Streams* i *Vert.x* kako bi podržao reaktivne obrasce.
- Koncept dodataka pruža integraciju s raznim tehnologijama, okruženjima i uslugama. Dodaci olakšavaju konfiguraciju i upotrebu vanjskih alata i usluga, kao što su baze podataka, sustavi za razmjenu poruka, sigurnost, mikro profiliranje i mnogi drugi.
- *Quarkus* ima mogućnost generiranja nativne izvršne datoteke koja se izravno izvršava na serveru bez potrebe za JVM-om što omogućuje još bolju performansu, manju potrošnju memorije i manji resursni otisak.

Korištena verzija *Quarkus* web okvira u ovom radu *je 2.16.4.Final*.

3.2. Golang

Golang (*Go*) je jezik visoke razine, statički tipizirani, kompiliran i proceduralan programski jezik koji je dizajniran za jednostavnost, efikasnost i skalabilnost. *Go* je dizajniran da bude jednostavan za učenje i upotrebu. Ima čistu i izražajnu sintaksu s malo ključnih riječi, što olakšava pisanje i čitanje koda. Osim toga, *Go* naglašava jednostavnost dizajna i izbjegavanje suvišnih složenosti. Vrlo je efikasan jezik u pogledu resursa. Ima ugrađenu podršku za konkurentnost i paralelizam putem *Go* rutina i kanala koji omogućuju upravljanje konkurentnim procesima, što je posebno korisno za razvoj skalabilnih i brzih aplikacija. Provjera tipova podataka se vrši tijekom kompilacije, što pomaže u otkrivanju grešaka u ranoj fazi razvojnog procesa. Tipovi su jednostavni i jasni, što olakšava razumijevanje i održavanje koda. Dolazi s velikom standardnom bibliotekom koja obuhvaća mnoge uobičajene funkcionalnosti. Biblioteka sadrži pakete za rad s mrežom, enkripciju, obradu nizova, upravljanje datotekama, testiranje i mnogo više. Koristi se za razvoj različitih vrsta aplikacija, uključujući web poslužitelje, mikro usluge, alate za infrastrukturu, razvoj aplikacija za umjetnu inteligenciju i još mnogo toga [18]. Korištena verzija *Go-a* je *go1.20.2 darwin/arm64*. Korištena verzija *Go Docker* slike je *Golang*:1.19.

3.2.1. Gin

Gin je brz, lagan i fleksibilan radni okvir za razvoj web aplikacija u programskom jeziku *Go*. *Gin* je izgrađen na osnovama jezika *Go* i pruža jednostavnu i učinkovitu sintaksu za izgradnju skalabilnih web aplikacija.

Neke ključne značajke *Gin* radnog okvira [19]:

- Dizajniran je da bude iznimno brz i efikasan što se postiže kroz optimizaciju u svojoj implementaciji, kao što su minimalna refleksija i brza pretvorba podataka koji ulaze i izlaze iz aplikacije.
- Ima jednostavnu sintaksu koja olakšava pisanje i čitanje koda. On pruža intuitivne načine za definiranje ruta, upravljača i međuprogramske podrška.
- Pruža fleksibilnost u pogledu konfiguracije i prilagodbe. Može se lako prilagoditi definiranje ruta, dodati ili ukloniti međuprogramska podrška, i kontrolirati kako se aplikacija ponaša.
- Podržava koncept međuprogramske podrške koji omogućava dodavanje funkcionalnosti poput logiranja, autentikacije, autorizacije ili validacije podataka na razini zahtjeva. Ovo olakšava modularnost i ponovno korištenje koda.
- Ima veliku i aktivnu zajednicu koja pruža mnoge dodatne pakete i modul za različite potrebe. Ima dobro dokumentiranu biblioteku s primjerima i vodičima koji pomažu u brzom učenju i implementaciji.

Korištena verzija *Gin* web okvira u ovom radu je *1.9.0.*

3.2.2. Echo

Echo je popularan radni okvir za razvoj web aplikacija u programskom jeziku *Go*. Nekoliko ključnih značajki koje čine *Echo* posebnim [20]:

- Ima jednostavnu i čistu sintaksu koja olakšava pisanje i održavanje koda. Nudi intuitivne metode za definiranje ruta, rukovatelja i međuprogramske podrške.
- Optimiziran je za brzinu izvršavanja. Koristi efikasne metode za rukovanje HTTP zahtjevima i odgovorima, što rezultira visokom performansom web aplikacija. Također podržava "*hot code reloading*", što omogućava brze iteracije tijekom razvoja.
- Fleksibilan je radni okvir koji omogućava prilagodbu prema potrebama projekta. Omogućava jednostavno dodavanje i konfiguriranje međuprogramske podrške za proširenje funkcionalnosti aplikacije. Podržava grupiranje ruta i organizaciju rukovatelja na logičan način.
- Podržava koncept međuprogramske podrške koji olakšava implementaciju funkcionalnosti kao što su autentikacija, autorizacija, logiranje, kompresija odgovora i još neke. Međuprogramska podrška se može globalno primijeniti na sve rute ili samo na određene rute, pružajući fleksibilnost u obradi zahtjeva.

- Ima aktivnu zajednicu programera koja pruža podršku i kontinuirano doprinosi razvoju radnog okvira. Također ima i dobro dokumentiranu biblioteku i mnogo primjera koji olakšavaju učenje i implementaciju.

Korištena verzija *Echo* web okvira je *4.10.2*.

3.3. Python

Python je jezik visokog nivoa, objektno orijentirani, interpretirani i dinamički tipiziran programski jezik koji se ističe svojom jednostavnošću, čitljivošću i fleksibilnošću. Stvoren je s ciljem da bude lako razumljiv i intuitivan za programiranje. *Python* se koristi u različitim područjima, kao što su web razvoj, znanstveno istraživanje, analiza podataka, umjetna inteligencija, automatizacija i mrežno programiranje. *Python* ima čistu i preglednu sintaksu koja olakšava čitanje i pisanje koda. Naglašava čitljivost i održavanje, koristeći indentaciju umjesto zagrada za definiranje blokova koda. *Python* dolazi s obiljem ugrađenih modula i funkcionalnosti u svojoj standardnoj biblioteci. Ova biblioteka obuhvaća širok spektar zadataka, poput obrade nizova, rad s datotekama, mrežnog programiranja, upravljanja bazama podataka, jednostavnog stvaranja GUI-ja i još neke. Izvršava se putem prevoditelja, što omogućava interaktivno isprobavanje i brzi razvoj aplikacija. Nema potrebe za prevođenjem izvornog koda u strojni jezik, već se *Python* izvodi izravno iz izvornog koda. *Python* je dinamički tipizirani jezik, što znači da se vrste varijabli ne moraju eksplicitno određivati prilikom deklaracije. Tipovi se automatski određuju prilikom dodjele vrijednosti varijablama [21]. Važno je napomenuti da su *Python* 2.x i *Python* 3.x značajno različite verzije, te da *Python* 2.x više nije aktivno razvijan i održavan od siječnja 2020. godine. Preporučuje se korištenje *Python*a 3.x za sve nove projekte, jer donosi najnovije značajke, poboljšane performanse i ispravke sigurnosnih propusta. *Python* je popularan izbor među programerima zbog svoje jednostavnosti, fleksibilnosti i široke primjene. Nudi alate i mogućnosti za rješavanje raznih programerskih izazova, bilo da se radi o malim skriptama ili složenim aplikacijama.

Korištena verzija *Python*-a u lokalnom razvoju je 3.11.2. Korištena verzija *Python Docker* slike je *python:3.11-slim-buster*.

3.3.1. FastAPI

FastAPI je moderan, brz i visokoučinkovit radni okvir za razvoj web aplikacija u programskom jeziku *Python*. Ističe se sljedećim značajkama [22]:

- Vrlo brz i učinkovit radni okvir zahvaljujući korištenju *ASGI* (Asynchronous Server Gateway Interface) protokola i *Pydantic* paketa za modeliranje podataka. ASGI omogućava asinkrono izvršavanje i visoke performanse, dok *Pydantic* paket pruža brzu validaciju i serijalizaciju podataka.
- Koristi deklarativnu sintaksu, što olakšava definiranje ruta i modela podataka. Koristi tipove podataka kako bi automatski generirao dokumentaciju, upute za korištenje i provjeru ulaznih podataka. Time se smanjuje potreba za ručnim pisanjem dokumentacije i povećava produktivnost programera.
- Temeljen je na *Python* tipovima podataka, što omogućava provjeru tipova tijekom izvođenja i automatsko generiranje dokumentacije. Time se mogućava sigurniji razvoj, smanjuje mogućnost greške i olakšava održavanje koda.
- Automatski generira interaktivnu dokumentaciju temeljenu na *OpenAPI* standardu. Ova dokumentacija pruža pregled dostupnih ruta, parametara, odgovora i omogućava isprobavanje API-ja izravno iz preglednika. Time se omogućava dokumentiranje i testiranje API-ja brzim i jednostavnim.
- Podržava asinkroni razvoj, što omogućava rukovanje više zahtjeva paralelno i efikasno korištenje resursa. Navedena značajka je korisna za razvoj aplikacija koje zahtijevaju visok broj korisnika i paralelno izvršavanje zadataka.

FastAPI je sve popularniji izbor za razvoj *Python* web aplikacija zbog svoje brzine, jednostavnosti i modernih značajki. Omogućuje programerima brži razvoj sigurnih i skalabilnih API-ja te pruža integraciju s drugim popularnim alatima kao što su *Tortoise ORM*, *SQLAlchemy* i druge.

3.3.2. Flask

Flask je lagani, fleksibilni i popularni radni okvir za razvoj web aplikacija u programskom jeziku *Python*.

Neke ključne značajke *Flask* radnog okvira [23]:

- Jednostavan i minimalistički web okvir koji nudi samo osnovne komponente potrebne za izgradnju web aplikacija, što ga čini laganijim i prilagodljivijim. *Flask* ne nameće stroge konvencije i omogućava programerima da sami oblikuju strukturu aplikacije.
- *Flask* je mikro okvir, što znači da pruža samo osnovne funkcionalnosti i ne dolazi s mnogo ugrađenih značajki. Time se omogućava programerima veću slobodu i fleksibilnost u odabiru i integraciji drugih biblioteka i alata prema potrebama projekta.

- Podržava *Jinja2* predložak koji omogućava programerima da generiraju dinamički HTML sadržaj. Predložak omogućava razdvajanje logike i prikaza, što olakšava održavanje koda i izradu privlačnih korisničkih sučelja.
- Modularan radni okvir za koji postoji velik broj ekstenzija dostupnih koje dodaju dodatne funkcionalnosti kao što su autentikacija, upravljanje bazama podataka, API razvoj, obrada obrazaca i još mnogo toga. Ove ekstenzije olakšavaju razvoj i smanjuju potrebu za pisanjem koda od nule.

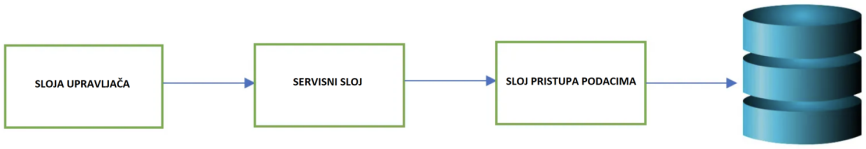
3.4. Arhitektura web usluge

Slojevi web usluga se koriste u organizaciji koda kod razvoja aplikacija radi odvajanja različitih odgovornosti i postizanja bolje modularnosti. Ovi slojevi često uključuju upravljač, servis i pristup podacima (engl. DAO - *Data Access Object*).

Upravljač je sloj koji obrađuje zahtjeve korisnika i upravlja protokom podataka unutar aplikacije. Ovaj sloj prima ulazne zahtjeve, obrađuje ih i generira odgovore. Upravljač je odgovoran za upravljanje korisničkim interakcijama, a često sadrži logiku za definiranje ruta, validaciju unosa, autorizaciju i slanje odgovora. Cilj upravljača je osigurati da zahtjevi budu pravilno usmjereni i da se podaci prenesu na odgovarajuće slojeve.

Servisni sloj sadrži poslovnu logiku aplikacije. Ovaj sloj obično izvršava složenije operacije i koordinira rad različitih DAO objekata. Servisni sloj prima zahtjeve od upravljača i obavlja odgovarajuće operacije kako bi ispunio poslovne zahtjeve. Na primjer, servisni sloj može provjeravati autentičnost korisnika, izračunavati kompleksne izračune, komunicirati s vanjskim sustavima itd. Servisni sloj pruža apstrakciju i centralizaciju poslovne logike, čime olakšava ponovnu upotrebu i testiranje.

Sloj pristupa podacima je odgovoran za komunikaciju s izvorom podataka, kao što su baze podataka, datoteke ili vanjski API-ji. On pruža metode za izvršavanje operacija na podacima, kao što su dohvaćanje, spremanje, ažuriranje i brisanje. Ovaj sloj je odgovoran za izvođenje upita nad podacima i mapiranje rezultata na objekte koji se koriste u aplikaciji. Također pruža apstrakciju pristupa podacima i olakšava zamjenu izvora podataka bez utjecaja na druge slojeve aplikacije. Segmentirana aplikacija po slojevima se može vizualno predočiti kao na slici 3.1..



Slika 3.1. *Arhitektura aplikacije [24]*

3.5. Vrste testnih scenarija

Sve aplikacije korištene u radu imaju jedinstveni API koji koristi REST protokole i JSON format. U sloju upravljača na zadanim rutama su implementirani navedeni testni slučajevi:

- **GET api/v1/test/case1** – testiranje brzine odziva aplikacije prilikom JSON serijalizacije podataka.
- **POST api/v1/test/case2** – testiranje brzine odziva aplikacije prilikom JSON deserijalizacije i serijalizacije podataka.
- **GET api/v1/test/case3/{number}** – testiranje brzine odziva aplikacije prilikom računanja *n*-tog broja Fibonacci-jevog red i serijalizacije podataka u JSON objekt.
- **POST api/v1/test/case4** – testiranje brzine odziva aplikacije prilikom JSON deserijalizacije podataka, spremanja objekta osobe koja sadrži ime, prezime, godinu rođenja i identifikacijski broj koji je i primarni ključ u bazi podataka, čitanje iste osobe prethodno spremljene u bazu po primarnom ključu, brisanje prethodno spremljene osobe te prosljeđivanje podataka o spremljenoj osobi u odgovoru na upit koji prolazi JSON serijalizaciju. Komunikacija s bazom se odvija preko tradicionalnih SQL upita.
- **POST api/v1/test/case5** – testiranje brzine odziva aplikacije prilikom JSON deserijalizacije podataka, spremanja objekta osobe koja sadrži ime, prezime, godinu rođenja i identifikacijski broj koji je i primarni ključ u bazi podataka, čitanje iste osobe prethodno spremljene u bazu po primarnom ključu, brisanje prethodno spremljene osobe te prosljeđivanje podataka o spremljenoj osobi u odgovoru na upit koji prolazi JSON serijalizaciju. Ovaj testni slučaj ima višu razinu apstrakcije nego case4, jer koristi ORM (engl. Object-Relation Mapper) preko čijih autogeneriranih funkcija pristupa bazi podataka, bez potrebe pisanja SQL koda.

3.6. *Docker* i *Docker*-compose

Docker je otvorena platforma koja omogućuje pakiranje, distribuciju i izvođenje aplikacija unutar kontejnera. Kontejneri su izolirana okruženja koja sadrže aplikacijski kôd, izvršne datoteke, biblioteke i sve druge potrebne komponente za pokretanje aplikacije. *Docker* omogućuje lako reproduciranje aplikacija i osigurava da se aplikacija izvodi dosljedno na različitim sustavima [25].

Docker Compose je alat koji omogućuje definiranje i upravljanje više *Docker* kontejnera kao jedinstvene aplikacije. Pomoću *Docker Compose-a* možete definirati sve potrebne kontejnere, njihove konfiguracije, mrežne veze i druge resurse potrebne za pokretanje aplikacije. Ovo je korisno kada aplikacija sastoji od više servisa ili komponenti koje trebaju međusobno komunicirati [26].

Kombinacijom *Docker*-a i *Docker Compose-a* programerima je omogućeno jednostavno pakiranje aplikacija u kontejnere i njihovo pokretanje na različitim okruženjima bez brige o kompatibilnosti s operativnim sustavom. *Docker* kontejnerima omogućena je izolacija aplikacija, olakšano upravljanje resursima i konzistentno okruženje za izvođenje aplikacija. *Docker Compose*-om olakšano je upravljanje složenim aplikacijama s više servisa i omogućeno je brzo pokretanje i skaliranje aplikacija. U radu je korišten *Docker* za generiranje slike glavne aplikacije, čija se datoteka pokreće preko *Docker Compose* datoteke zajedno sa slikom *PostgreSQL* baze podataka koja se koristi u četvrtom i petom testnom slučaju. Korištena je i *Docker* ekstenzija *Resource Usage* kako bi bilo praćeno korištenje CPU (engl. Central Processing Unit) i memorije prilikom testiranja. Korištena naredba za izradu *Docker* slike *Docker-compose build* kojom se povlače sve biblioteke koje su korištene u aplikaciji unutar *Docker* kontejnera. Za pokretanje aplikacije unutar *Docker* kontejnera u ovom se radu koristi *Docker-compose up* kojom se prvo pokrene *PostgreSQL* baza podataka unutar kontejnera, nakon toga se kreiraju tablice koje su korištene u aplikaciji poslije čega se pokrene glavna aplikacija koja je povezana s bazom podataka.

3.7. Apache JMeter

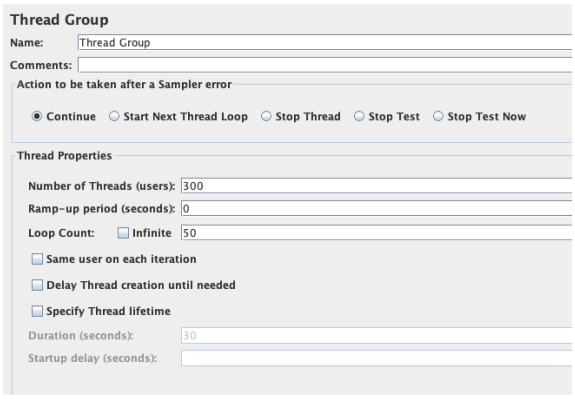
Apache JMeter je popularan alat za testiranje performansi i opterećenje web aplikacija. On omogućuje programerima i QA inženjerima da simuliraju veliki broj korisnika koji pristupaju web aplikaciji kako bi provjerili njezinu skalabilnost, pouzdanost i performanse. *Apache JMeter* omogućuje generiranje opterećenja na web aplikaciji simulirajući paralelne korisnike i mjeri vrijeme odziva i performanse. Može se testirati broj korisnika koji se istovremeno pristupa aplikaciji, brzina odgovora, vremena odziva, opterećenje poslužitelja i druge performanse. *Apache*

JMeter podržava testiranje različitih vrsta protokola kao što su HTTP, HTTPS, FTP, SOAP, REST, JDBC, JMS itd. Ovo omogućuje testiranje performansi različitih vrsta aplikacija i servisa. *Apache JMeter* omogućuje korisnicima da generiraju složene scenarije testiranja koji uključuju različite korake, postavke podataka i interakcije s aplikacijom. Scenariji se mogu konfigurirati za testiranje specifičnih funkcionalnosti, simulaciju stvarnih korisničkih aktivnosti i mjerenje performansi. *Apache JMeter* prikuplja detaljne rezultate performansi tijekom testiranja, uključujući vremena odziva, broj zahtjeva, uspješnost, greške itd. Ovi rezultati se mogu vizualizirati u obliku grafova i izvještaja za detaljnu analizu performansi aplikacije. *Apache JMeter* omogućuje distribuirano testiranje, što znači da se opterećenje može podijeliti između više računala kako bi se simuliralo veliko broj korisnika. Sve navedeno omogućuje testiranje skalabilnosti aplikacije i mjerenje performansi u stvarnom svijetu. *Apache JMeter* je moćan alat za testiranje performansi koji omogućuje programerima da identificiraju i riješe probleme s performansama u njihovim web aplikacijama [27]. Korištena verzija *JMetera* je 5.6.2.

4. REZULTATI TESTIRANJA

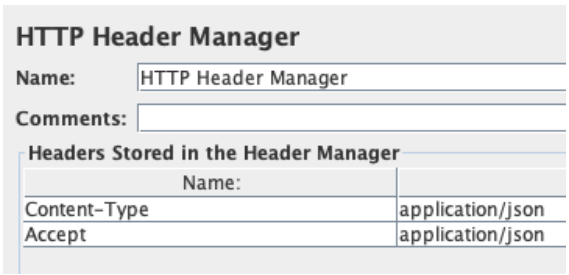
4.1. Osnovne postavke JMeter alata

U sklopu testiranja koje se provodilo na računalu *MacBook Pro* sa čipom *Apple M1 Pro*, 32 GB RAM memorije (engl. Random Access Memory) i operativnim sustavom *Ventura 13.5*, u svakom testnom scenariju, aplikacija je opterećena sa 300 korisnika, koji je ekvivalentan broju niti korištenih za slanje upita aplikaciji. Razdoblje povećanja je postavljeno na 0 sekundi i govori koliko je vremena potrebno do aktivacije maksimalnog broja zadanih niti za testiranje. Svaka nit izvršava upit po 50 puta, što znači da će aplikacija biti testirana sa 15 000 upita (Slika 4.1.).



Slika 4.1. *Postavke broja niti i upita testiranja*

Upiti sadrže zaglavlje sa parametrima *Content-Type* i *Accept* koji imaju vrijednost *application/JSON* jer je komunikacija u JSON formatu (Slika 4.2.). Aplikacije imaju dostupno po 5 jezgri u *Docker* kontejneru pa se iskoristivost resursa procesora kreće od 0 do 500%.



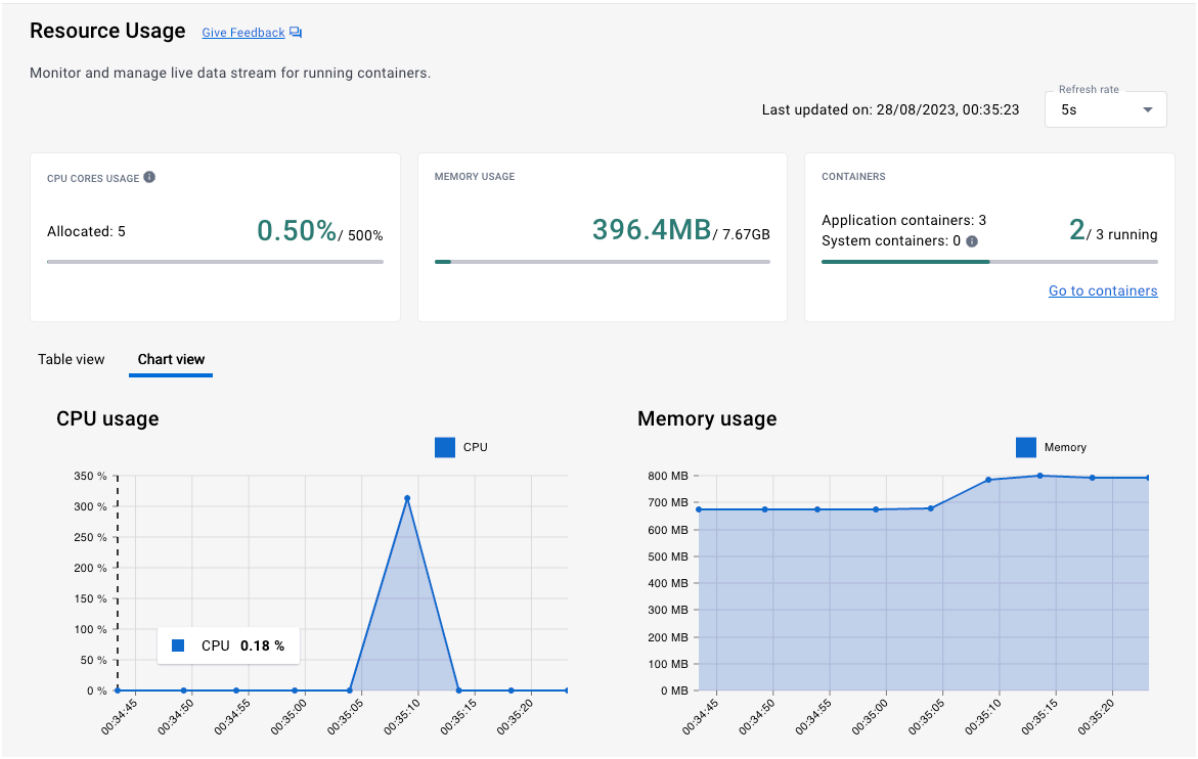
Slika 4.2. *Postavka zaglavlja upita testiranja*

4.2. Prvi testni scenarij u kojem je testirana brzina odziva aplikacije prilikom JSON serijalizacije podataka

U prvom testnom scenariju poslan je upit **GET api/v1/test/case1** te testiranje brzine odziva aplikacije prilikom JSON serijalizacije podataka.

4.2.1. Prvi testni scenarij web okvira Spring Boot

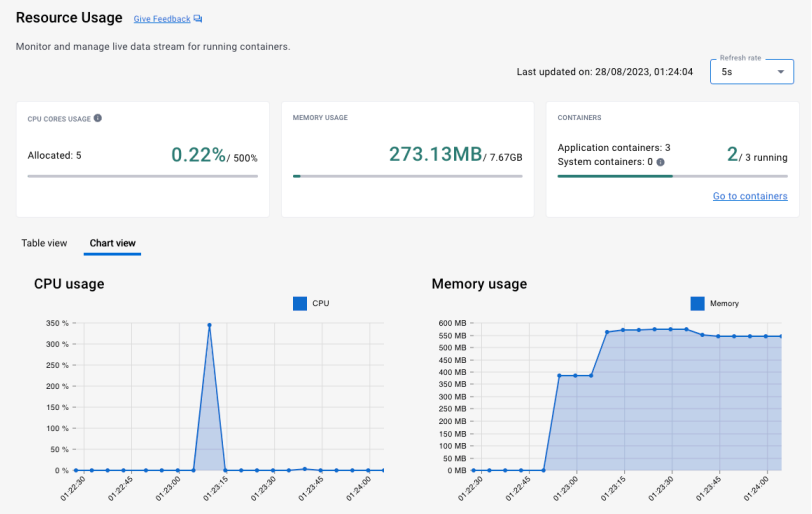
Spring Boot aplikacija se pokreće u *Docker* kontejneru na portu 8080. Početak testiranja je u 00:35 sati. Korištenje resursa procesora i memorije je vidljivo na slici 4.3.. Najveće korištenje resursa procesora *Spring Boot* aplikacije u prvom testnom scenariju je oko 305%. Aplikacija je prilikom testiranja koristila oko 800 MB, dok je u stanju mirovanja koristila 700 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 100 MB.



Slika 4.3. *Iskoristivost resursa prilikom testiranja prvog testnog scenarija web okvira Spring Boot*

4.2.2. Prvi testni scenarij web okvira Quarkus

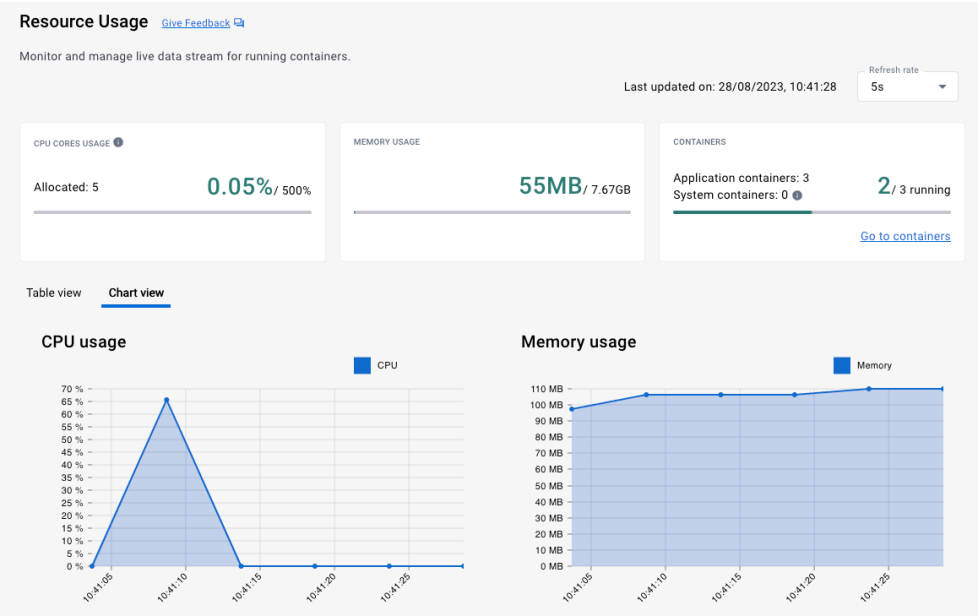
Quarkus aplikacija se pokreće u *Docker* kontejneru na portu 8081. Početak testiranja je u 01:23 sati. Korištenje resursa procesora i memorije je vidljivo na slici 4.4.. Najveće korištenje resursa procesora *Quarkus* aplikacije u prvom testnom scenariju je oko 350% što je i najviše u usporedbi s ostalim aplikacijama prilikom testiranja prvog testnog scenarija. Aplikacija je prilikom testiranja koristila oko 575 MB, dok je u stanju mirovanja koristila 400 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 175 MB što je najviše u usporedbi s ostalim aplikacijama prilikom testiranja prvog testnog scenarija.



Slika 4.4. *Iskoristivost resursa prilikom testiranja prvog testnog scenarija web okvira Quarkus*

4.2.3. Prvi testni scenarij web okvira Gin

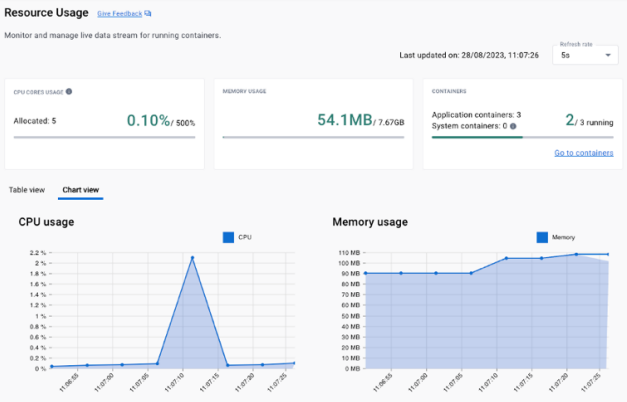
Gin aplikacija se pokreće u *Docker* kontejneru na portu 8082. Početak testiranja je u 10:41 sati. Korištenje resursa procesora i memorije je vidljivo na slici 4.5.. Najveće korištenje resursa procesora *Gin* aplikacije u prvom testnom scenariju je oko 65%. Aplikacija je prilikom testiranja koristila oko 110 MB, dok je u stanju mirovanja koristila 100 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 10 MB što je najmanje u usporedbi s ostalim aplikacijama prilikom testiranja prvog testnog scenarija.



Slika 4.5. *Iskoristivost resursa prilikom testiranja prvog testnog scenarija web okvira Gin*

4.2.4. Prvi testni scenarij web okvira Echo

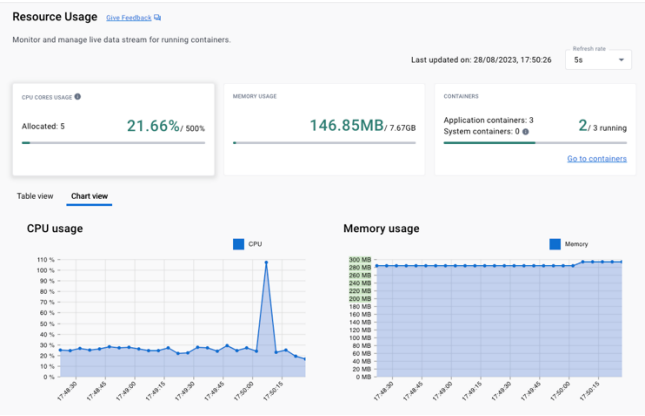
Echo aplikacija se pokreće u *Docker* kontejneru na portu 8083. Početak testiranja je u 11:07 sati. Korištenje resursa procesora i memorije je vidljivo na slici 4.6.. Najveće korištenje resursa procesora *Echo* aplikacije u prvom testnom scenariju je oko 2% što je i najmanje u usporedbi s ostalim aplikacijama prilikom testiranja prvog testnog scenarija. Aplikacija je prilikom testiranja koristila oko 110 MB, dok je u stanju mirovanja koristila 90 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 20 MB.



Slika 4.6. *Iskoristivost resursa prilikom testiranja prvog testnog scenarija web okvira Echo*

4.2.5. Prvi testni scenarij web okvira FastAPI

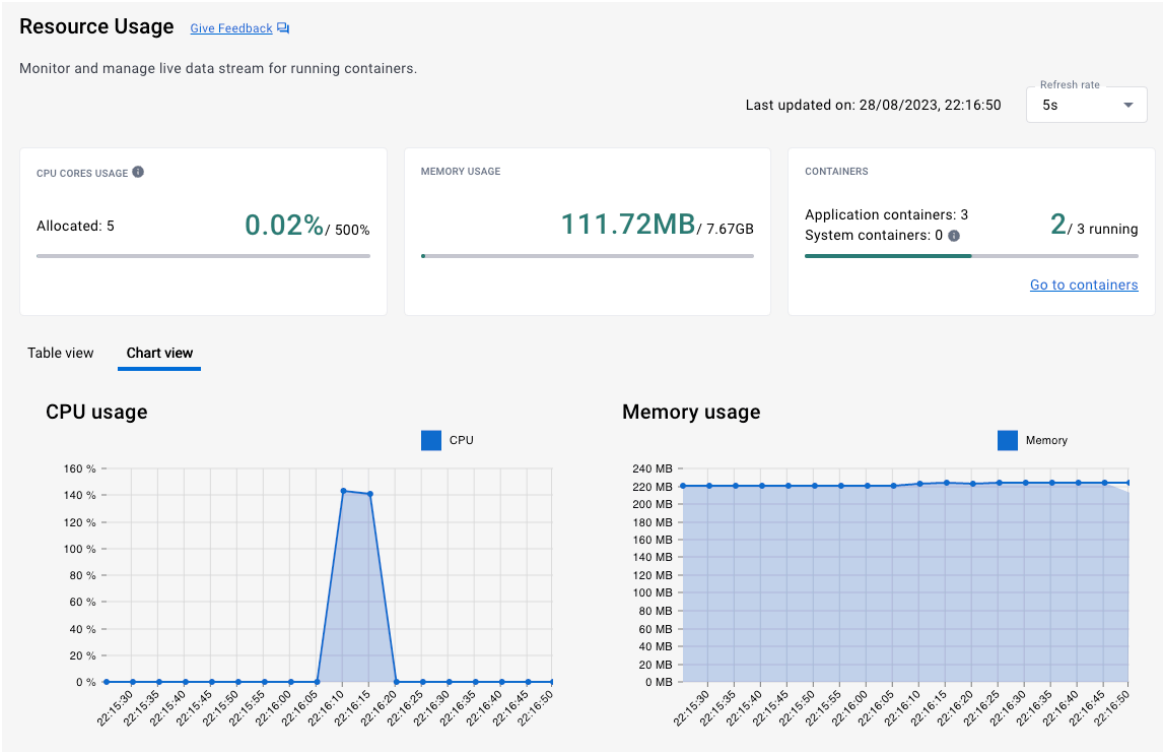
FastAPI aplikacija se pokreće u *Docker* kontejneru na portu 8084. Početak testiranja je u 17:50 sati. Korištenje resursa procesora i memorije je vidljivo na slici 4.7. Najveće korištenje resursa procesora *FastAPI* aplikacije u prvom testnom scenariju je oko 110% što je i najviše u usporedbi s ostalim aplikacijama kod testiranja prvog testnog scenarija. Aplikacija je prilikom testiranja koristila oko 295 MB, dok je u stanju mirovanja koristila 280 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 15 MB.



Slika 4.7. *Iskoristivost resursa prilikom testiranja prvog testnog scenarija web okvira FastAPI*

4.2.6. Prvi testni scenarij web okvira Flask

Flask aplikacija se pokreće u *Docker* kontejneru na portu 8085. Početak testiranja je u 22:16 sati. Korištenje resursa procesora i memorije je vidljivo na slici 4.8.. Najveće korištenje resursa procesora *Flask* aplikacije u prvom testnom scenariju je oko 140% što je i najviše u usporedbi s ostalim aplikacijama prilikom testiranja prvog testnog scenarija. Aplikacija je prilikom testiranja koristila oko 230 MB, dok je u stanju mirovanja koristila 220 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 10 MB što je najmanje u usporedbi s ostalim aplikacijama prilikom testiranja prvog testnog scenarija.



Slika 4.8. *Iskoristivost resursa prilikom testiranja prvog testnog scenarija web okvira Flask*

4.2.7. Usporedba metričkih rezultata prvog testnog scenarija

U prvom testnom scenariju najmanje resursa procesora i memorije koriste *Golang* web okviri, dok najviše koriste *Java* web okviri. *Echo* koristi najmanje resursa procesora, 2% od ukupno 500% što znači da najbolje optimizira korištenje resursa procesora pri JSON serijalizaciji, no i postotak može biti mali jer je uzorkovanje Resource Usage ekstenzije manje od brzine obrade svih upita prvog testnog slučaja, dok *Quarkus* koristi najviše, 350%. *Echo* koristi najmanje resursa memorije, u stanju mirovanja oko 90 MB dok u fazi testiranja koristi 20 MB više, ukupno oko 110 MB. *Spring Boot* koristi najviše resursa memorije, u stanju mirovanja 700 MB, dok u fazi testiranja

koristi 100 MB više, ukupnu oko 800 MB. Najmanju razliku korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja imaju *Gin* i *Flask*, po 10 MB. Najveću razliku korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja ima *Quarkus,* 175 MB.

Najkraće prosječno vrijeme izvršavanja pojedinačnog upita imaju *Golang* web okviri; *Gin* 24 milisekunde i *Echo* 27 milisekundi. Najduže prosječno vrijeme izvršavanja pojedinačnog upita imaju *Python* web okviri; *FastAPI* 77 milisekundi i *Flask* 149 milisekundi. Median vrijednost izvršavanja pojedinačnog upita je najmanja kod *Spring Boot* web okvira, 16 milisekundi, a prate ga *Golang* web okviri, *Echo* sa 17 milisekundi i *Gin* sa 21 milisekundu. Najveća medijan vrijednost izvršavanja pojedinačnog upita je kod *Flask* web okvira, 87 milisekundi. Najmanji postotak neizvršenih upita po testiranju ima *Spring Boot*, 0.03% dok najveći imaju *Golang* web okviri, oko 1.6 %. Najmanje operacija po sekundi ima *Flask*, svega 1419.9 operacija po sekundi, dok najviše operacija po sekundi imaju *Golang* web okviri; *Gin* 9265 operacija po sekundi i *Echo* 7637.5 operacija po sekundi. Najmanja prosječna veličina odgovora na upit je kod *Quarkus*-a, 54 bita, a najveća kod *Golang* web okvira, oko 201 bit. Ukupno trajanje izvršavanja testa je najsporije kod *Flask* web okvira, 10 sekundi, dok je najbrže kod *Golang* web okvira i iznosti 1 sekundu (tablica 4.1.).

Metrički rezultati prikazani u tablicama 4.1., 4.2., 4.3., 4.4., 4.5. na kraju svakog testnog scenarija prikazuju redom [28]:

- Web okvir – korišteni web okvir
- Broj uzoraka – broj upita korišten u testu performansi
- Prosjek – aritmetička sredina vremena svih odgovora na upit; u milisekundama
- Median – vrijeme niza rezultata testa koji pokazuje potrebno vrijeme izvršavanja za polovicu upita; za izvršavanje ostatka upita potrebno je najmanje isto toliko vremena; u milisekundama
- Greška – prikazuje postotak neizvršenih upita po testiranju; u postotcima
- Propusnost – prikazuje broj zahtjeva koji se obrađuju po sekundi; u broju obrađenih zahtjevima po sekundi
- Prosječni bitovi – prosječna veličina odgovora na upit; u bitovima
- Trajanje – vrijeme trajanja testiranja; u minutama i sekundama

Tablica 4.1. *Metrički rezultati testiranja performansi prvog testnog slučaja*

Web okvir	Broj uzoraka	Prosjek / milisekunda	Median / milisekunda	Greška / %	Propusnost / operacija/sekunda	Prosjek / bit	Trajanje / minuta:sekunda
Spring Boot	15000	51	16	0.03	3958.8/sec	164.7	0:03
Quarkus	15000	54	34	1.30	4922.9/sec	54	0:03
Gin	15000	24	21	1.60	9265.0/sec	201	0:01
Echo	15000	27	17	1.59	7637.5/sec	201.4	0:01
FastAPI	15000	77	67	1.28	3820.7/sec	195.2	0:03
Flask	15000	149	87	0.56	1419.9/sec	199.5	0:10

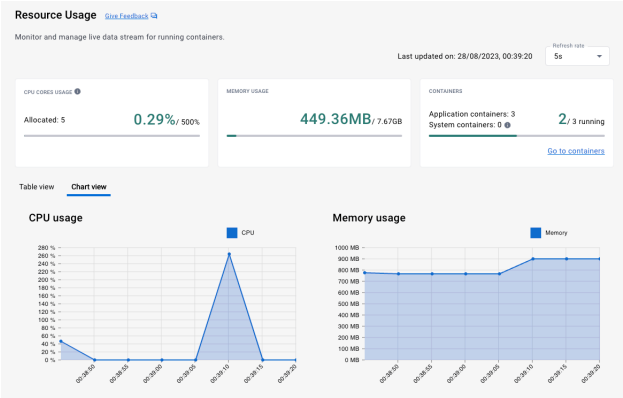
4.3. Drugi testni scenarij u kojem je testirana brzina odziva aplikacije prilikom JSON deserijalizacije i serijalizacije podataka

U prvom testnom scenariju poslan je upit **POST api/v1/test/case2** te testiramo brzinu odziva aplikacije prilikom JSON deserijalizacije i serijalizacije podataka.

Tijelo poruke :*{"name":"Ime"}*

4.3.1. Drugi testni scenarij web okvira Spring Boot

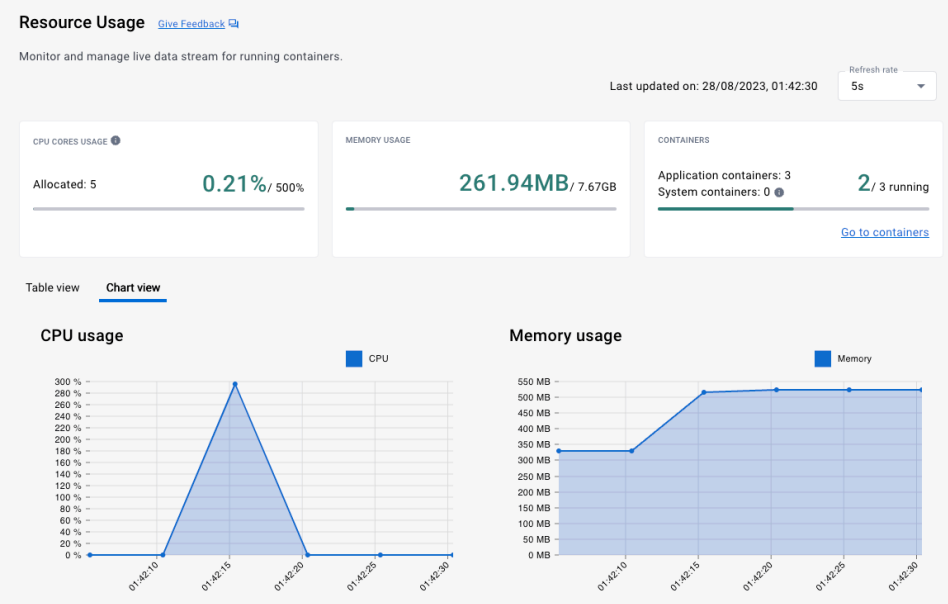
Spring Boot aplikacija se pokreće u *Docker* kontejneru na portu 8080. Početak testiranja je u 00:39 sati. Korištenje resursa procesora i memorije je vidljivo na slici 4.9.. Najveće korištenje resursa procesora *Spring Boot* aplikacije u drugom testnom scenariju je oko 280%. Aplikacija je prilikom testiranja koristila oko 900 MB, dok je u stanju mirovanja koristila 800 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 100 MB.



Slika 4.9. *Iskoristivost resursa prilikom testiranja drugog testnog scenarija web okvira Spring Boot*

4.3.2. Drugi testni scenarij web okvira Quarkus

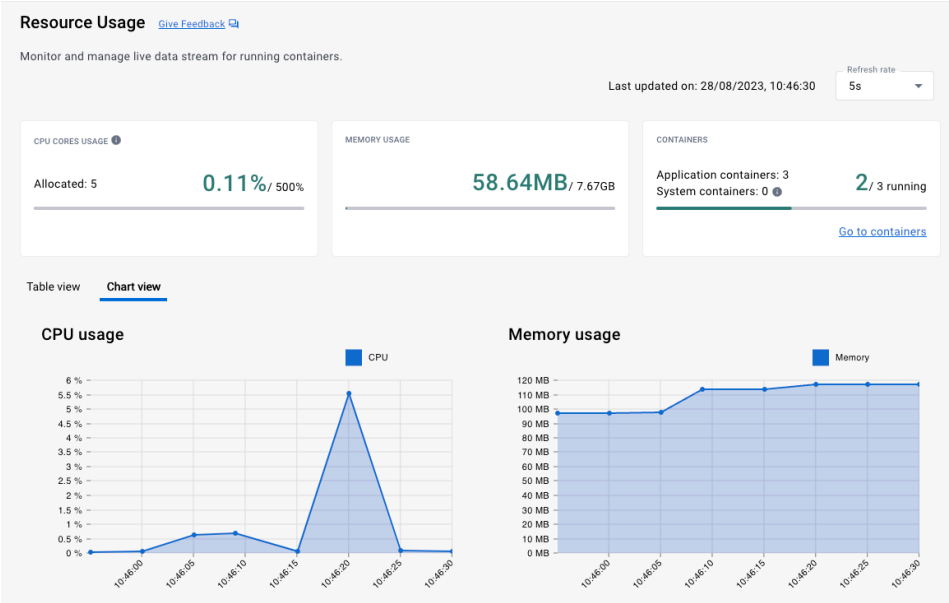
Quarkus aplikacija se pokreće u *Docker* kontejneru na portu 8081. Početak testiranja je u 01:42 sati. Korištenje resursa procesora i memorije je vidljivo na slici 4.10.. Najveće korištenje resursa procesora *Quarkus* aplikacije u drugom testnom scenariju je oko 300% što je i najviše u usporedbi s ostalim aplikacijama prilikom testiranja drugog testnog scenarija. Aplikacija je prilikom testiranja koristila oko 525 MB, dok je u stanju mirovanja koristila oko 330 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 195 MB što je najviše u usporedbi s ostalim aplikacijama prilikom testiranja drugog testnog scenarija.



Slika 4.10. *Iskoristivost resursa prilikom testiranja drugog testnog scenarija web okvira Quarkus*

4.3.3. Drugi testni scenarij web okvira Gin

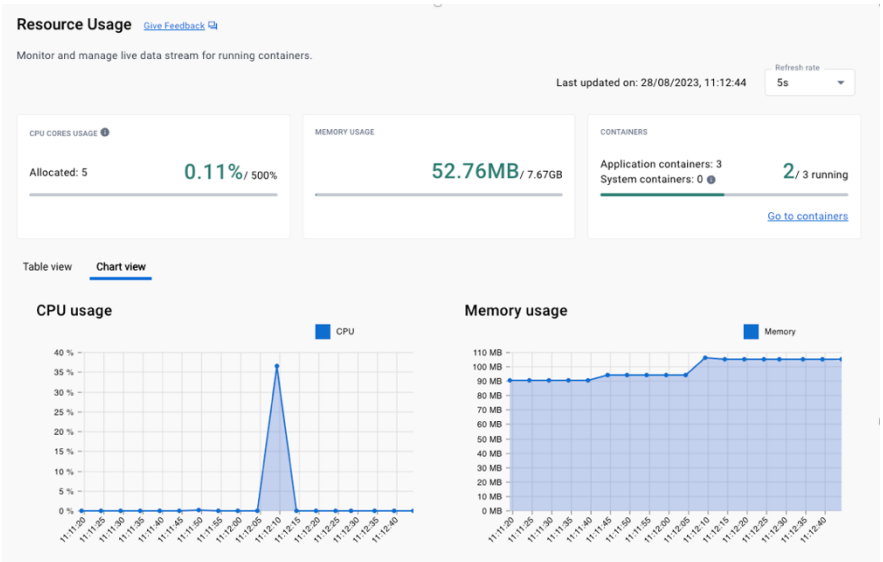
Gin aplikacija se pokreće u *Docker* kontejneru na portu 8082. Početak testiranja je u 10:46 sati. Korištenje resursa procesora i memorije je vidljivo na slici 4.11.. Najveće korištenje resursa procesora *Gin* aplikacije u drugom testnom scenariju je oko 5.5% što je najmanja iskoristivost resursa procesora u usporedbi s ostalim aplikacijama prilikom testiranja drugog testnog scenarija. Aplikacija je prilikom testiranja koristila oko 110 MB, dok je u stanju mirovanja koristila oko 100 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 10 MB što je najmanje u usporedbi s ostalim aplikacijama prilikom testiranja drugog testnog scenarija.



Slika 4.11. *Iskoristivost resursa prilikom testiranja drugog testnog scenarija web okvira Gin*

4.3.4. Drugi testni scenarij web okvira Echo

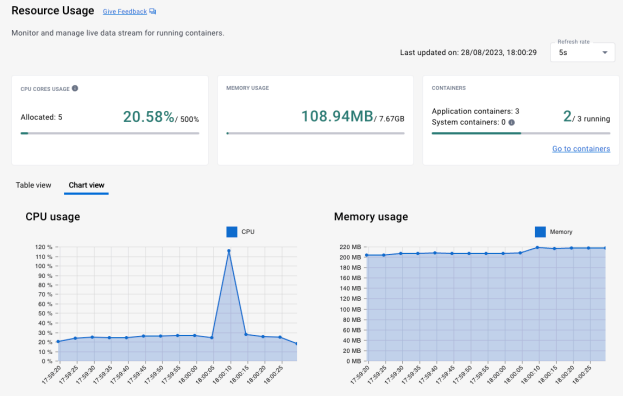
Echo aplikacija se pokreće u *Docker* kontejneru na portu 8083. Početak testiranja je u 11:12 sati. Korištenje resursa procesora i memorije je vidljivo na slici 4.12.. Najveće korištenje resursa procesora *Echo* aplikacije u drugom testnom scenariju je oko 35%. Aplikacija je prilikom testiranja koristila oko 105 MB, dok je u stanju mirovanja koristila oko 90 MB memorije, što je najmanja iskoristivost resursa memorije u ovom testnom scenariju. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 15 MB.



Slika 4.12. *Iskoristivost resursa prilikom testiranja drugog testnog scenarija web okvira Echo*

4.3.5. Drugi testni scenarij web okvira FastAPI

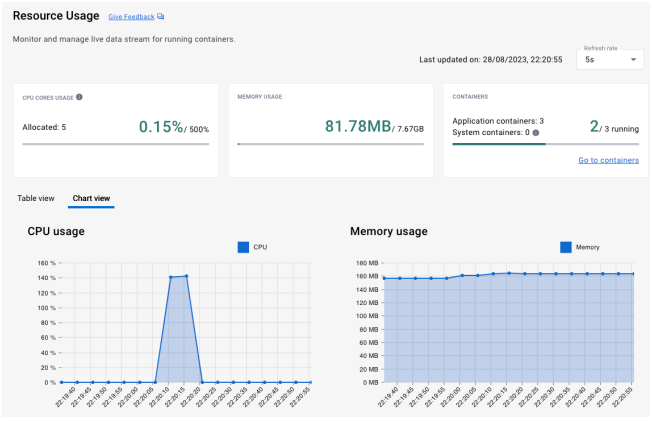
FastAPI aplikacija se pokreće u *Docker* kontejneru na portu 8084. Početak testiranja je u 18:00 sati. Korištenje procesora i memorije je vidljivo na slici 4.13.. Najveće korištenje resursa procesora *FastAPI* aplikacije u drugom testnom scenariju je oko 110%. Aplikacija je prilikom testiranja koristila oko 220 MB, dok je u stanju mirovanja koristila oko 205 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 15 MB.



Slika 4.13. *Iskoristivost resursa prilikom testiranja drugog testnog scenarija web okvira FastAPI*

4.3.6. Drugi testni scenarij web okvira Flask

Flask aplikacija se pokreće u *Docker* kontejneru na portu 8085. Početak testiranja je u 22:20 sati. Korištenje procesora i memorije je vidljivo na slici 4.14.. Najveće korištenje resursa procesora *Flask* aplikacije u drugom testnom scenariju je oko 140%. Aplikacija je prilikom testiranja koristila oko 165 MB, dok je u stanju mirovanja koristila oko 155 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 10 MB što je najmanje u usporedbi s ostalim aplikacijama prilikom testiranja drugog testnog scenarija.



Slika 4.14. *Iskoristivost resursa prilikom testiranja drugog testnog scenarija web okvira Flask*

4.3.7. Usporedba metričkih rezultata drugog testnog scenarija

U drugom testnom scenariju najmanje resursa procesora i memorije koriste *Golang* web okviri, dok najviše koriste *Java* web okviri. *Gin* koristi najmanje resursa procesora, 5.5% od ukupno 500%, dok *Quarkus* koristi najviše, 300%. *Echo* koristi najmanje resursa memorije, u stanju mirovanja oko 90 MB dok u fazi testiranja koristi 15 MB više, ukupno oko 105 MB. *Spring Boot* koristi najviše resursa memorije, u stanju mirovanja 800 MB, dok u fazi testiranja koristi 100 MB više, ukupnu oko 900 MB. Najmanju razliku korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja imaju *Gin* i *Flask*, po 10 MB. Najveću razliku korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja ima *Quarkus,* 195 MB.

Najkraće prosječno vrijeme izvršavanja pojedinačnog upita imaju *Golang* web okviri; *Gin* 39 milisekunde i *Echo* 38 milisekundi. Najduže prosječno vrijeme izvršavanja imaju *Python* web okviri; *FastAPI* 78 milisekundi i *Flask* 171 milisekundi. Median vrijednost izvršavanja upita je najmanja kod *Spring Boot* web okvira, 19 milisekundi, a prate ga *Golang* web okviri, *Echo* sa 21 milisekundu i *Gin* sa 26 milisekundi. Najveća medijan vrijednost izvršavanja upita je kod *Flask* web okvira, 93 milisekunde. Najmanji postotak neizvršenih upita po testiranju ima *Spring Boot*, 0.17% dok najveći ima *Echo* web okvir, 1.81 %. Najmanje operacija po sekundi ima *Flask*, svega 1005.9 operacija po sekundi, dok najviše operacija po sekundi imaju *Golang* web okviri; *Gin* 5197.5 operacija po sekundi i *Echo* 5372.5 operacija po sekundi. Najmanja prosječna veličina odgovora na upit je kod *Quarkus*-a, 156.7 bita, a najveća kod *Golang* web okvira i *Flask-a*, oko 205 bita. Ukupno trajanje izvršavanja testa je najsporije kod *Flask* web okvira, 14 sekundi, dok je najbrže kod *Golang* web okvira i iznosti 2 sekundu (tablica 4.2.).

Tablica 4.2. *Metrički rezultati testiranja performansi drugog testnog scenarija*

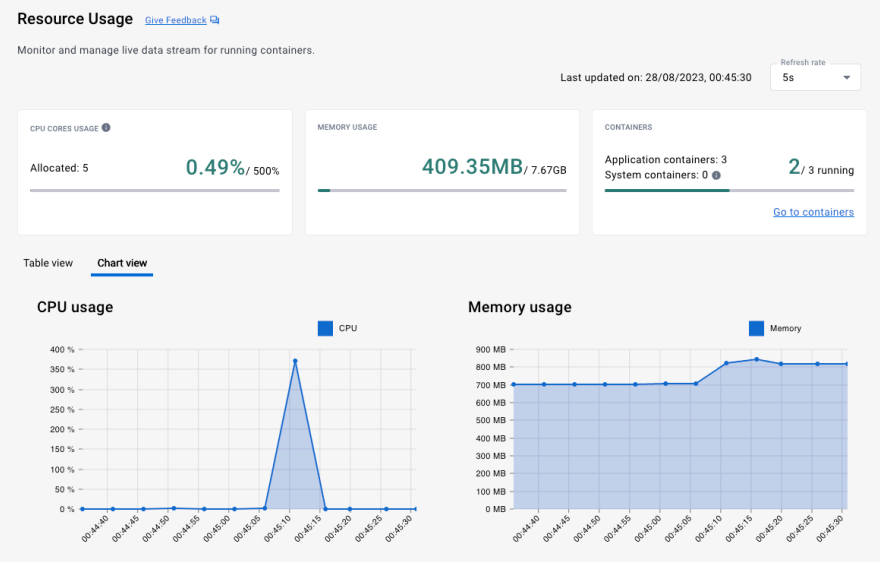
Web okvir	**Broj uzoraka**	**Prosjek / milisekunda**	**Median / milisekunda**	**Greška / %**	**Propusnost / operacija/sekunda**	**Prosjek / bit**	**Trajanje / minuta:sekunda**
Spring Boot	15000	54	19	0.17	4046.4/sec	172.1	0:03
Quarkus	15000	56	39	1.73	4507.2/sec	156.7	0:03
Gin	15000	39	26	1.56	5197.5/sec	203.9	0:02
Echo	15000	38	21	1.81	5372.5/sec	210.8	0:02
FastAPI	15000	78	73	1.48	3784.1/sec	204	0:03
Flask	15000	171	93	0.67	1005.9/sec	206.1	0:14

4.4. Treći testni scenarij u kojem je testirana brzina odziva aplikacije prilikom računanja n-tog broja Fibonacci-jevog reda i serijalizacije podataka u JSON objekt

U prvom testnom scenariju poslan je upit GET api/v1/test/case3/28 te testiramo brzinu odziva aplikacije prilikom računanja n-tog broja Fibonacci-jevog red i serijalizacije podataka u JSON objekt.

4.4.1. Treći testni scenarij web okvira Spring Boot

Spring Boot aplikacija se pokreće u *Docker* kontejneru na portu 8080. Početak testiranja je u 00:45 sati. Korištenje resursa procesora i memorije je vidljivo na slici 4.15.. Najveće korištenje resursa procesora *Spring Boot* aplikacije u trećem testnom scenariju je oko 375%. Aplikacija je prilikom testiranja koristila oko 840 MB, dok je u stanju mirovanja koristila oko 700 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 140 MB.

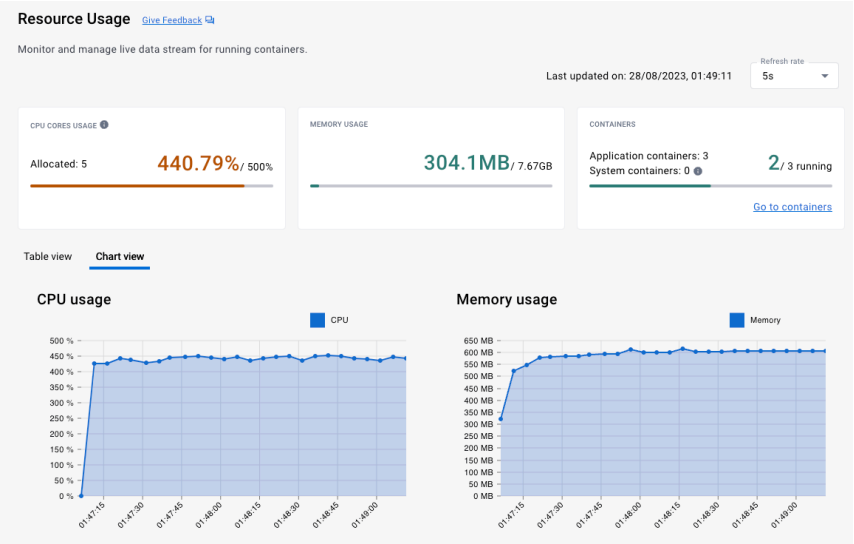


Slika 4.15. *Iskoristivost resursa prilikom testiranja trećeg testnog scenarija web okvira Spring Boot*

4.4.2. Treći testni scenarij web okvira Quarkus

Quarkus aplikacija se pokreće u *Docker* kontejneru na portu 8081. Početak testiranja je u 01:47 sati. Korištenje resursa procesora i memorije je vidljivo na slici 4.16.. Najveće korištenje resursa procesora *Quarkus* aplikacije u trećem testnom scenariju je oko 450% što je i najviše u usporedbi s ostalim aplikacijama prilikom testiranja trećeg testnog scenarija. Aplikacija je

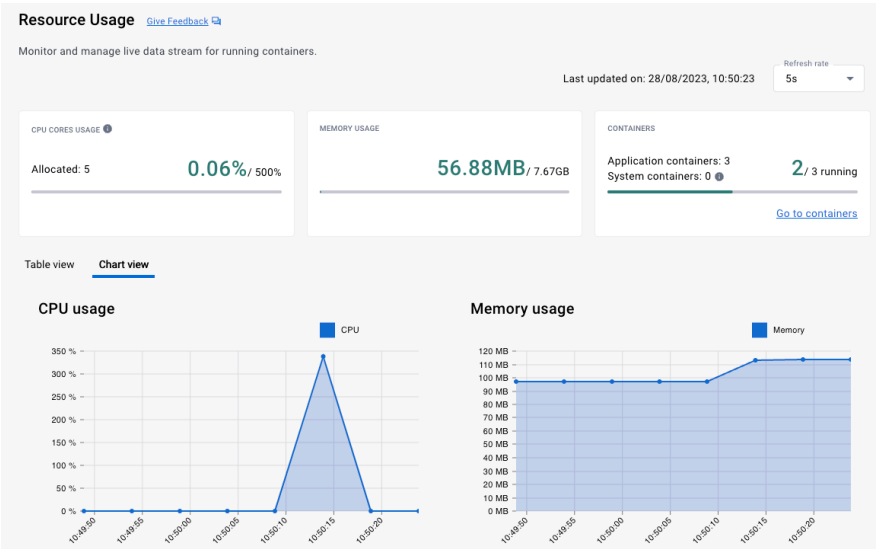
prilikom testiranja koristila oko 600 MB, dok je u stanju mirovanja koristila oko 300 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 300 MB što je najviše u usporedbi s ostalim aplikacijama prilikom testiranja trećeg testnog scenarija.



Slika 4.16. *Iskoristivost resursa prilikom testiranja trećeg testnog scenarija web okvira Quarkus*

4.4.3. Treći testni scenarij web okvira Gin

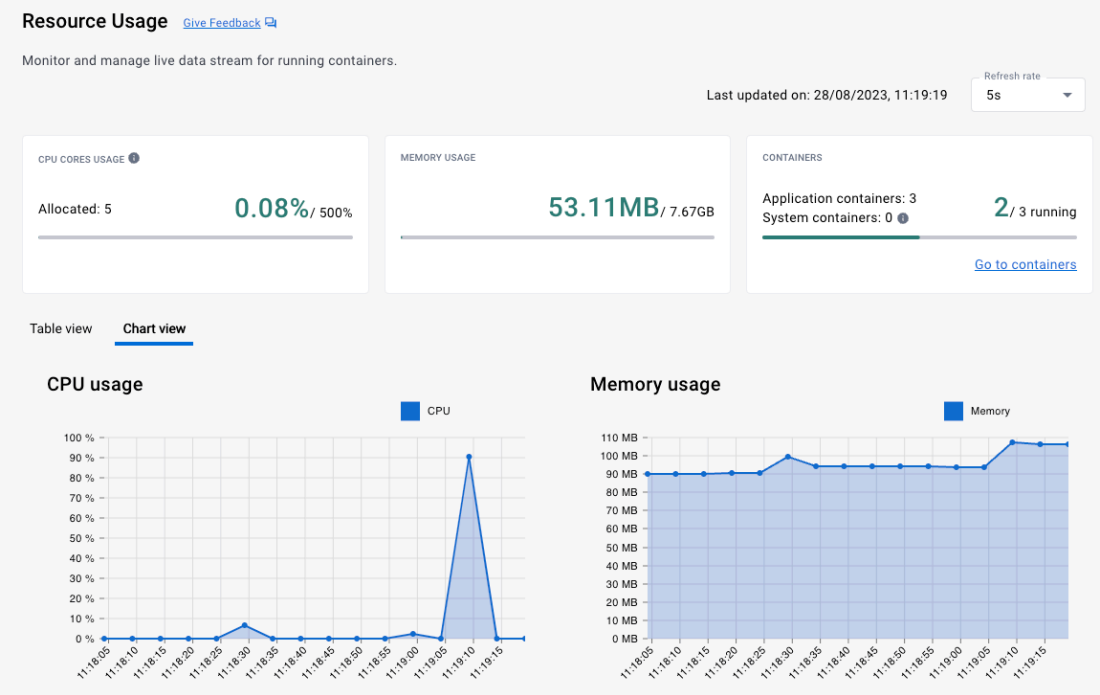
Gin aplikacija se pokreće u *Docker* kontejneru na portu 8082. Početak testiranja je u 10:50 sati. Korištenje resursa procesora i memorije je vidljivo na slici 4.17.. Najveće korištenje resursa procesora *Gin* aplikacije u trećem testnom scenariju je oko 350%. Aplikacija je prilikom testiranja koristila oko 150 MB, dok je u stanju mirovanja koristila oko 100 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 50 MB.



Slika 4.17. *Iskoristivost resursa prilikom testiranja trećeg testnog scenarija web okvira Gin*

4.4.4. Treći testni scenarij web okvira Echo

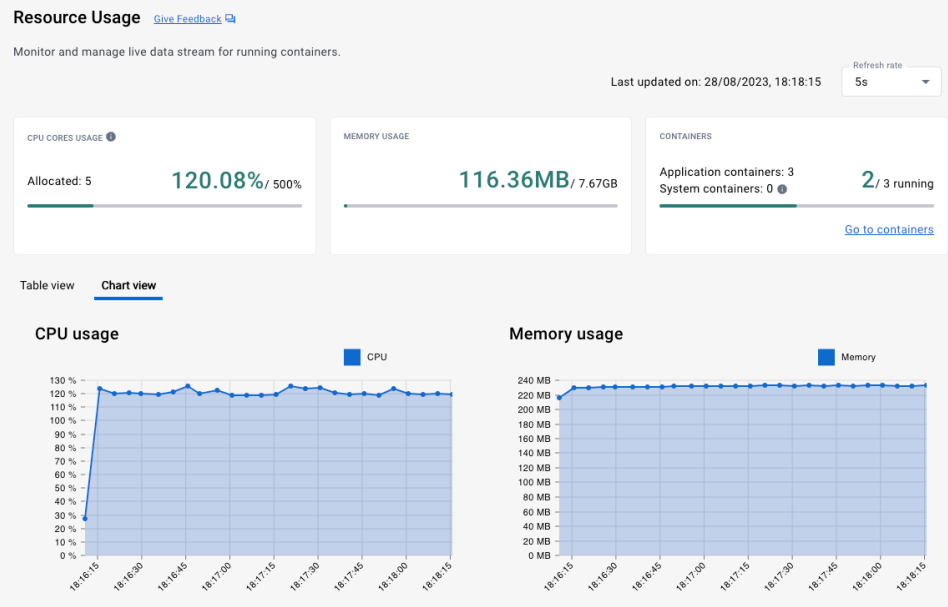
Echo aplikacija se pokreće u *Docker* kontejneru na portu 8083. Početak testiranja je u 11:19 sati. Korištenje resursa procesora i memorije je vidljivo na slici 4.18.. Najveće korištenje resursa procesora *Echo* aplikacije u trećem testnom scenariju je oko 90% što je najmanje u usporedbi s ostalim aplikacijama prilikom testiranja trećeg testnog scenarija. Aplikacija je prilikom testiranja koristila oko 90 MB, dok je u stanju mirovanja koristila oko 105 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 15 MB što je najmanje u usporedbi s ostalim aplikacijama prilikom testiranja trećeg testnog scenarija.



Slika 4.18. *Iskoristivost resursa prilikom testiranja trećeg testnog scenarija web okvira Echo*

4.4.5. Treći testni scenarij web okvira FastAPI

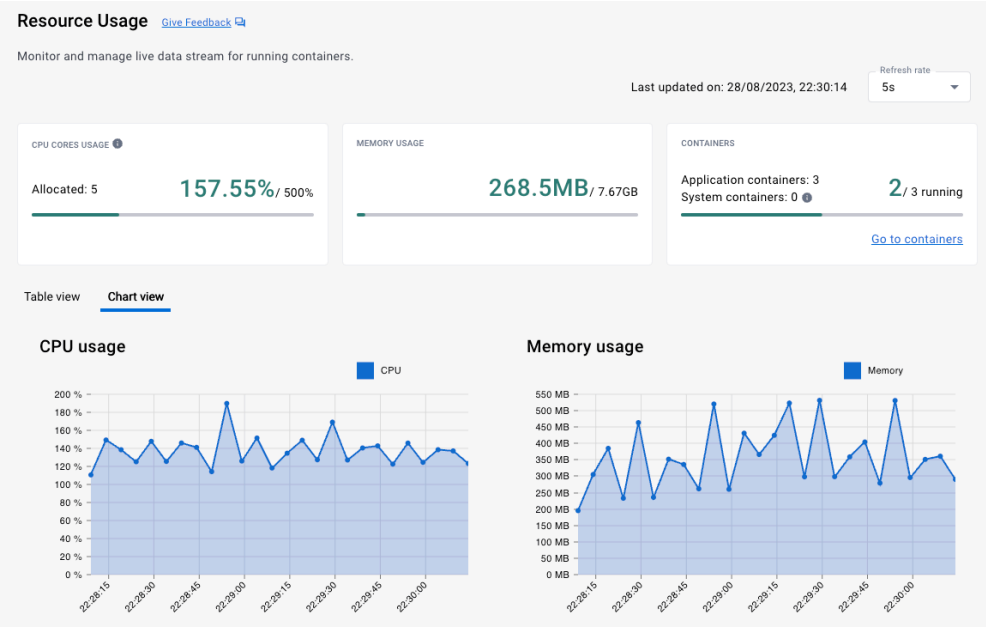
FastAPI aplikacija se pokreće u *Docker* kontejneru na portu 8084. Početak testiranja je u 18:16 sati. Korištenje resursa procesora i memorije je vidljivo na slici 4.19.. Najveće korištenje resursa procesora *FastAPI* aplikacije u trećem testnom scenariju je oko 125%. Aplikacija je prilikom testiranja koristila oko 230 MB, dok je u stanju mirovanja koristila oko 215 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 15 MB što je najmanje u usporedbi s ostalim aplikacijama prilikom testiranja trećeg testnog scenarija.



Slika 4.19. *Iskoristivost resursa prilikom testiranja trećeg testnog scenarija web okvira FastAPI*

4.4.6. Treći testni scenarij web okvira Flask

Flask aplikacija se pokreće u *Docker* kontejneru na portu 8085. Početak testiranja je u 22:28 sati. Korištenje resursa procesora i memorije je vidljivo na slikama 4.20.. Najveće korištenje resursa procesora *Flask* aplikacije u trećem testnom scenariju je oko 190%. Aplikacija je prilikom testiranja koristila oko 540 MB, dok je u stanju mirovanja koristila oko 200 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 340 MB.



Slika 4.20. *Iskoristivost resursa prilikom testiranja trećeg testnog scenarija web okvira Flask*

4.4.7. Usporedba rezultata trećeg testnog scenarija

U trećem testnom scenariju najmanje resursa procesora i memorije koristi *Echo* web okvir, dok najviše koriste *Java* web okviri. *Gin* koristi najmanje resursa procesora, 90% od ukupno 500%, dok *Quarkus* koristi najviše, 450%. *Echo* koristi najmanje resursa memorije, u stanju mirovanja oko 90 MB dok u fazi testiranja koristi 15 MB više, ukupno oko 105 MB. *Spring Boot* koristi najviše resursa memorije, u stanju mirovanja 700 MB, dok u fazi testiranja koristi 140 MB više, ukupnu oko 840 MB. Najmanju razliku korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja imaju *Echo* i *FastAPI*, po 15 MB. Najveću razliku korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja ima *Quarkus,* 300 MB kroz duži vremenski period.

Najkraće prosječno vrijeme izvršavanja pojedinačnog upita imaju *Golang* web okviri; *Gin* 78 milisekundi i *Echo* 80 milisekundi. Najduže prosječno vrijeme izvršavanja imaju *Python* web okviri; *FastAPI* 12395 milisekundi i *Flask* 8605 milisekundi. Median vrijednost izvršavanja upita je najmanja kod *Spring Boot* web okvira, 62 milisekunde, a prate ga *Golang* web okviri, *Echo* sa 97 milisekundi i *Gin* sa 103 milisekunde. Najveća medijan vrijednost izvršavanja upita je kod *FastAPI* web okvira, 12500 milisekundi. Najmanji postotak neizvršenih upita po testiranju ima *FastAPI*, 0.76% dok najveći ima *Flask* web okvir, 17.49 %. Najmanje operacija po sekundi ima *FastAPI*, svega 24 operacija po sekundi, dok najviše operacija po sekundi imaju *Golang* web okviri; *Gin* 2642.2 operacija po sekundi i *Echo* 2869.7 operacija po sekundi. Najmanja prosječna veličina odgovora na upit je kod *Quarkus*-a, 133.6 bita, a najveća kod *Flask-a*, 518.1 bit. Ukupno trajanje izvršavanja testa je najsporije kod *FastAPI* web okvira, 10 minuta i 26 sekundi, dok je najbrže kod *Golang* web okvira i iznosti 5 sekundi, *Spring Boot* je na drugom mjestu sa 7 sekundi.(tablica 4.3.).

Tablica 4.3. *Metrički rezultati testiranja performansi trećeg testnog slučaja*

Web okvir	**Broj uzoraka**	**Prosjek / milisekunda**	**Median / milisekunda**	**Greška / %**	**Propusnost / operacija/sekunda**	**Prosjek / bit**	**Trajanje / minuta:sekunda**
Spring Boot	15000	116	62	1.03	2187.2/sec	186.2	0:06
Quarkus	15000	3847	3807	1.07	76.02/sec	133.6	3:17
Gin	15000	103	78	1.04	2642.2/sec	184.4	0:05
Echo	15000	97	80	1.37	2869.7/sec	193.5	0:05
FastAPI	15000	12395	12500	0.76	24/sec	179.5	10:26
Flask	15000	8605	10534	17.49	30.8/sec	518.1	8:06

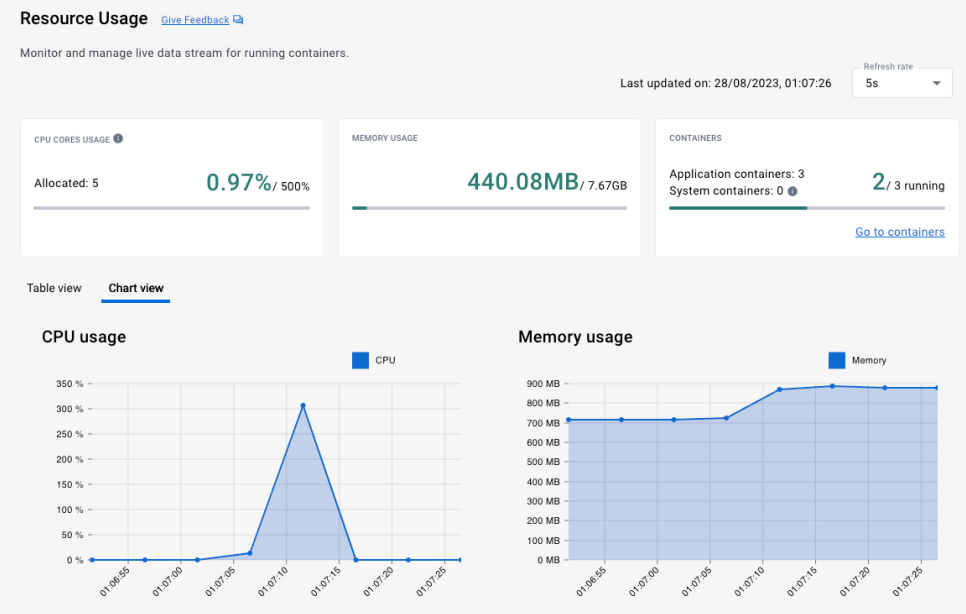
4.5. Četvrti testni scenarij u kojem je testirana brzina odziva prilikom JSON deserijalizacije i serijalizacije i komunikacije s bazom podataka preko tradicionalnih SQL upita

U prvom testnom scenariju poslan je upit **POST api/v1/test/case4** te testiramo brzinu odziva aplikacije prilikom JSON deserijalizacije podataka, spremanja objekta osobe koja sadrži ime, prezime, godinu rođenja i identifikacijski broj koji je i primarni ključ u bazi podataka, čitanje iste osobe prethodno spremljene u bazu po primarnom ključu, brisanje prethodno spremljene osobe te prosljeđivanje podataka o spremljenoj osobi u odgovoru na upit koji prolazi JSON serijalizaciju. Komunikacija s bazom se odvija preko tradicionalnih SQL upita.

Tijelo upita:*{"firstName":"Ime","lastName":"Prezime","yearOfBirth":2000}*

4.5.1. Četvrti testni scenarij web okvira Spring Boot

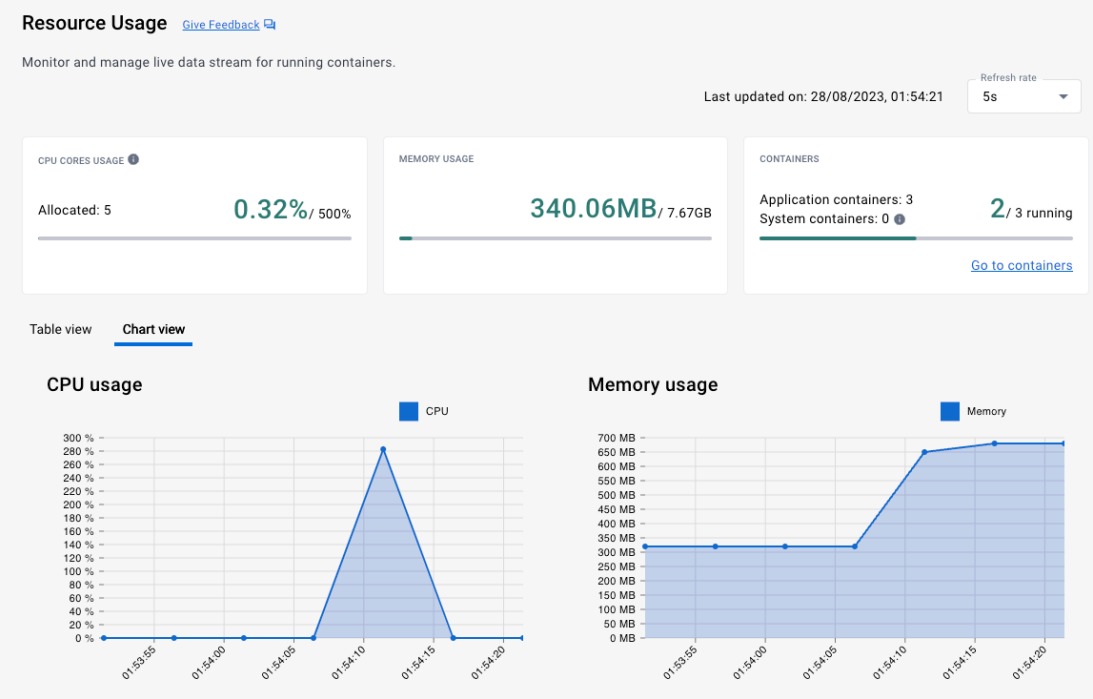
Spring Boot aplikacija se pokreće u *Docker* kontejneru na portu 8080. Početak testiranja je u 01:07 sati. Korištenje resursa procesora i memorije je vidljivo na slici 4.21.. Najveće korištenje resursa procesora *Spring Boot* aplikacije u četvrtom testnom scenariju je oko 305%. Aplikacija je prilikom testiranja koristila oko 900 MB, dok je u stanju mirovanja koristila oko 700 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 200 MB.



Slika 4.21. *Iskoristivost resursa prilikom testiranja četvrtog testnog scenarija web okvira Spring Boot*

4.5.2. Četvrti testni scenarij web okvira Quarkus

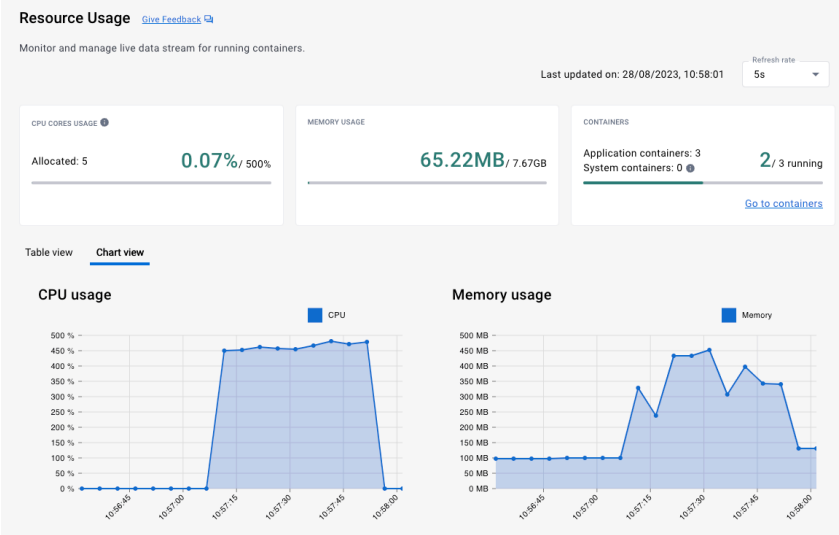
Quarkus aplikacija se pokreće u *Docker* kontejneru na portu 8081. Početak testiranja je u 01:54 sati. Korištenje resursa procesora i memorije je vidljivo na slici 4.22.. Najveće korištenje resursa procesora *Quarkus* aplikacije u četvrtom testnom scenariju je oko 280%. Aplikacija je prilikom testiranja koristila oko 675 MB, dok je u stanju mirovanja koristila oko 320 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 355 MB što je najviše u usporedbi s ostalim aplikacijama prilikom testiranja četvrtog testnog scenarija.



Slika 4.22. *Iskoristivost resursa prilikom testiranja četvrtog testnog scenarija web okvira Quarkus*

4.5.3. Četvrti testni scenarij web okvira Gin

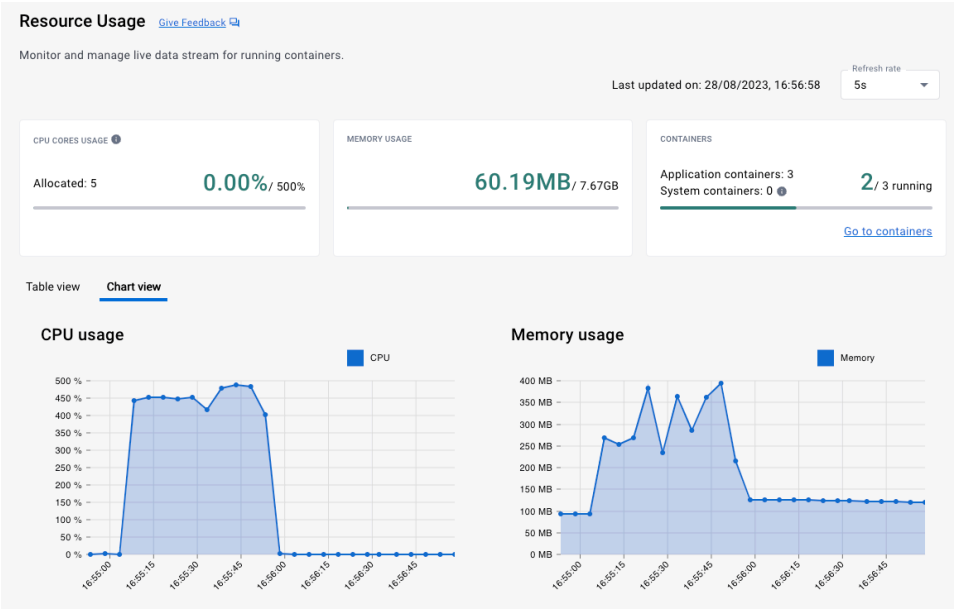
Gin aplikacija se pokreće u *Docker* kontejneru na portu 8082. Početak testiranja je u 10:57 sati. Korištenje resursa procesora i memorije je vidljivo na slici 4.23.. Najveće korištenje resursa procesora *Gin* aplikacije u četvrtom testnom scenariju je oko 480%. Aplikacija je prilikom testiranja koristila najviše oko 450 MB, dok je u stanju mirovanja koristila oko 100 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 350 MB.



Slika 4.23. *Iskoristivost resursa prilikom testiranja četvrtog testnog scenarija web okvira Gin*

4.5.4. Četvrti testni scenarij web okvira Echo

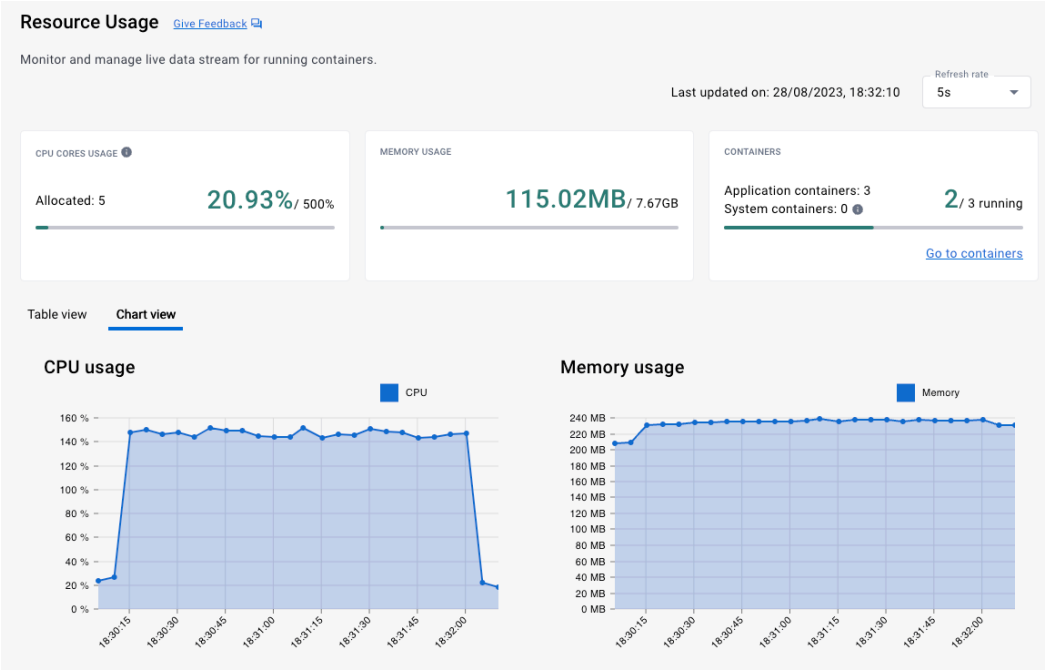
Echo aplikacija se pokreće u *Docker* kontejneru na portu 8083. Početak testiranja je u 16:55 sati. Korištenje resursa procesora i memorije je vidljivo na slici 4.24.. Najveće korištenje resursa procesora *Echo* aplikacije u četvrtom testnom scenariju je oko 490% što je i najviše u usporedbi s ostalim aplikacijama prilikom testiranja četvrtog testnog scenarija. Aplikacija je prilikom testiranja koristila najviše oko 400 MB, dok je u stanju mirovanja koristila oko 100 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 300 MB.



Slika 4.24. *Iskoristivost resursa prilikom testiranja četvrtog testnog scenarija web okvira Echo*

4.5.5. Četvrti testni scenarij web okvira FastAPI

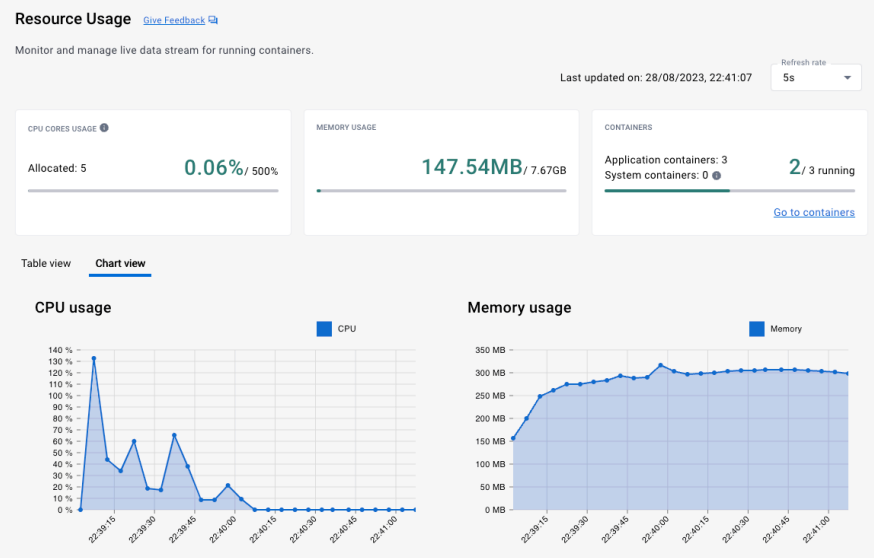
FastAPI aplikacija se pokreće u *Docker* kontejneru na portu 8084. Početak testiranja je u 18:30 sati. Korištenje resursa procesora i memorije je vidljivo na slici 4.25.. Najveće korištenje resursa procesora *FastAPI* aplikacije u četvrtom testnom scenariju je oko 150%. Aplikacija je prilikom testiranja koristila oko 240 MB, dok je u stanju mirovanja koristila oko 200 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 40 MB što je najmanje u usporedbi s ostalim aplikacijama prilikom testiranja četvrtog testnog scenarija.



Slika 4.25. *Iskoristivost resursa prilikom testiranja četvrtog testnog scenarija web okvira FastAPI*

4.5.6. Četvrti testni scenarij web okvira Flask

Flask aplikacija se pokreće u *Docker* kontejneru na portu 8085. Početak testiranja je u 22:39 sati. Korištenje resursa procesora i memorije je vidljivo na slici 4.26.. Najveće korištenje resursa procesora *Flask* aplikacije u četvrtom testnom scenariju je oko 130% što je i najmanje u usporedbi s ostalim aplikacijama prilikom testiranja četvrtog testnog scenarija.. Aplikacija je prilikom testiranja koristila oko 300 MB, dok je u stanju mirovanja koristila oko 150 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 150 MB.



Slika 4.26. *Iskoristivost resursa prilikom testiranja četvrtog testnog scenarija web okvira Flask*

4.5.7. Usporedba rezultata četvrtog testnog scenarija

U četvrtom testnom scenariju najmanje resursa procesora i memorije koristi *Python* web okviri. *Flask* koristi najmanje resursa procesora, 130% od ukupno 500%, dok *Echo* koristi najviše, 490%. *FastAPI* koristi najmanje resursa memorije, u stanju mirovanja oko 200 MB dok u fazi testiranja koristi 40 MB više, ukupno oko 240 MB. *Spring Boot* koristi najviše resursa memorije, u stanju mirovanja 700 MB, dok u fazi testiranja koristi 200 MB više, ukupnu oko 900 MB. Najmanju razliku korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja imaju *FastAPI*, 40 MB. Najveću razliku korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja ima *Quarkus,* 355 MB kroz duži vremenski period.

Najkraće prosječno vrijeme izvršavanja pojedinačnog upita imaju *Java* web okviri; *Spring Boot* 102 milisekunde i *Quarkus* 127 milisekundi. Najduže prosječno vrijeme izvršavanja imaju *Python* web okviri; *FastAPI* 2260 milisekundi i *Flask* 2591 milisekunda. Median vrijednost izvršavanja upita je najmanja kod *Spring Boot* web okvira, 51 milisekunde, a prate ga *Quarkus* web okvir sa 98 milisekundi. Najveća medijan vrijednost izvršavanja upita je kod *Flask* web okvira, 5011 milisekundi. Najmanji postotak neizvršenih upita po testiranju ima *Spring Boot*, 1.01% dok najveći imaju *Golang* web okviri, *Gin* 11.41% i *Echo* 11.83%. Najmanje operacija po sekundi ima *Flask*, svega 86.8 operacija po sekundi, dok najviše operacija po sekundi imaju *Java* web okviri; *Spring Boot* 2488.8 operacija po sekundi i *Quarkus* 2247.5 operacija po sekundi. Najmanja prosječna veličina odgovora na upit je kod *Quarkus*-a, 186.6 bita, a najveća kod *Flask-a*, 264 bit. Ukupno trajanje izvršavanja testa je najsporije kod *Flask* web okvira, 2 minute i 53 sekunde, dok je najbrže kod *Java* web okvira i iznosti 6 sekundi (tablica 4.3.).

Tablica 4.4. *Metrički podatci testiranja performansi četvrtog testnog slučaja*

Web okvir	Broj uzoraka	Prosjek / milisekunda	Median / milisekunda	Greška / %	Propusnost / operacija/sekunda	Prosjek / bit	Trajanje / minuta:sekunda
Spring Boot	15000	102	51	1.01	2488.8/sec	237.4	0:06
Quarkus	15000	127	98	1.12	2247.5/sec	186.6	0:06
Gin	15000	820	648	11.41	327.9/sec	232.2	0:45
Echo	15000	842	662	11.83	322.3/sec	253.2	0:46
FastAPI	15000	2260	2190	1.03	131.4/sec	241.7	1:53
Flask	15000	2591	5011	1.05	86.8/sec	264	2:53

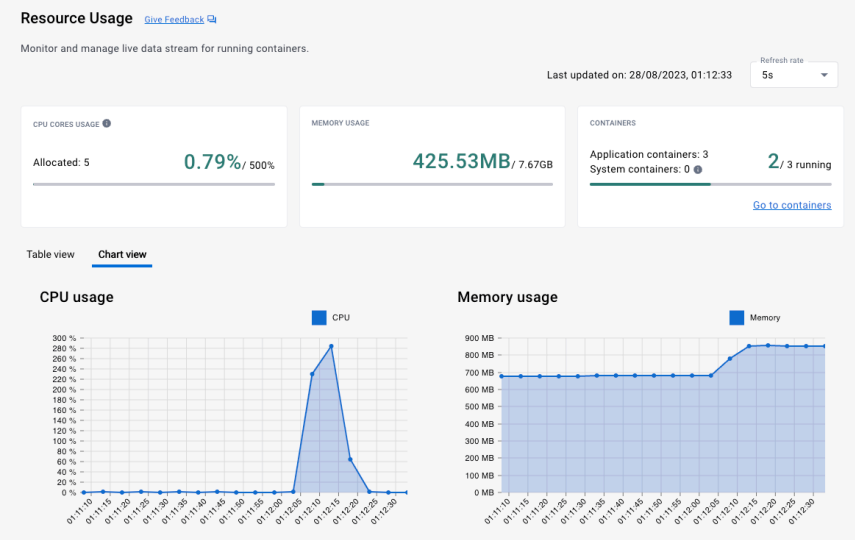
4.6. Peti testni scenarij je testirana brzina odziva prilikom JSON deserijalizacije i serijalizacije i komunikacije s bazom podataka korištenjem ORM-a

U prvom testnom scenariju poslan je upit **POST api/v1/test/case5** te testiramo brzinu odziva aplikacije prilikom JSON deserijalizacije podataka, spremanja objekta osobe koja sadrži ime, prezime, godinu rođenja i identifikacijski broj koji je i primarni ključ u bazi podataka, čitanje iste osobe prethodno spremljene u bazu po primarnom ključu, brisanje prethodno spremljene osobe te prosljeđivanje podataka o spremljenoj osobi u odgovoru na upit koji prolazi JSON serijalizaciju. Ovaj testni slučaj ima višu razinu apstrakcije nego četvrti testni slučaj, jer koristi ORM preko čijih autogeneriranih funkcija pristupa bazi podataka, bez potrebe pisanja SQL koda.

Tijelo upita:*{"firstName":"Ime","lastName":"Prezime","yearOfBirth":2000}*

4.6.1. Peti testni scenarij web okvira Spring Boot

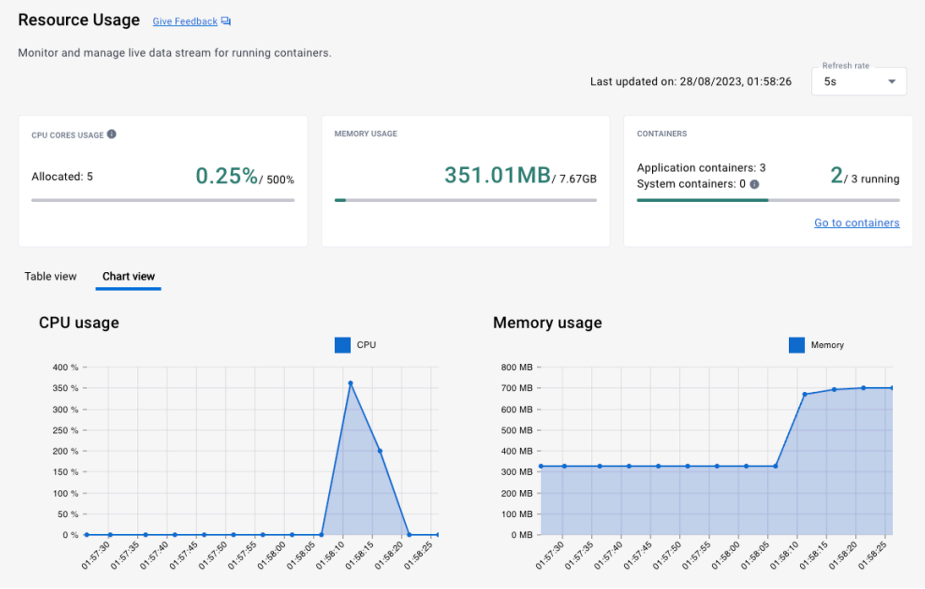
Spring Boot aplikacija se pokreće u *Docker* kontejneru na portu 8080. Početak testiranja je u 01:12 sati. Korištenje resursa procesora i memorije je vidljivo na slici 4.27.. Najveće korištenje resursa procesora *Spring Boot* aplikacije u petom testnom scenariju je oko 280%. Aplikacija je prilikom testiranja koristila oko 850 MB, dok je u stanju mirovanja koristila oko 700 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 150 MB.



Slika 4.27. *Iskoristivost resursa prilikom testiranja petog testnog scenarija web okvira Spring Boot*

4.6.2. Peti testni scenarij web okvira Quarkus

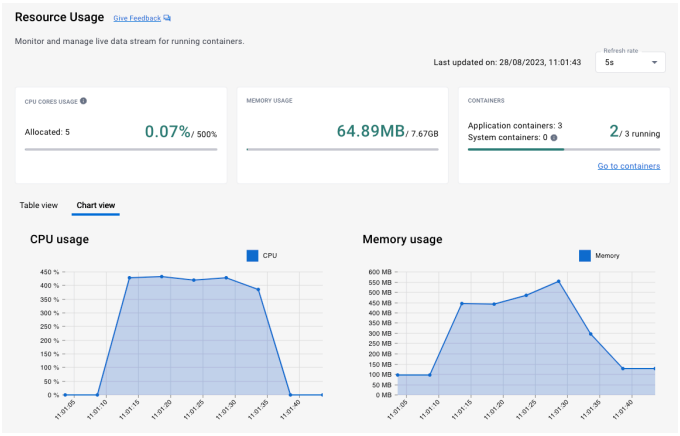
Quarkus aplikacija se pokreće u *Docker* kontejneru na portu 8081. Početak testiranja je u 01:58 sati. Korištenje resursa procesora i memorije je vidljivo na slici 4.28.. Najveće korištenje resursa procesora *Quarkus* aplikacije u petom testnom scenariju je oko 350%. Aplikacija je prilikom testiranja koristila oko 700 MB, dok je u stanju mirovanja koristila oko 300 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 400 MB što je najviše u usporedbi s ostalim aplikacijama prilikom testiranja petog testnog scenarija.



Slika 4.28. *Iskoristivost resursa prilikom testiranja petog testnog scenarija web okvira Quarkus*

4.6.3. Peti testni scenarij web okvira Gin

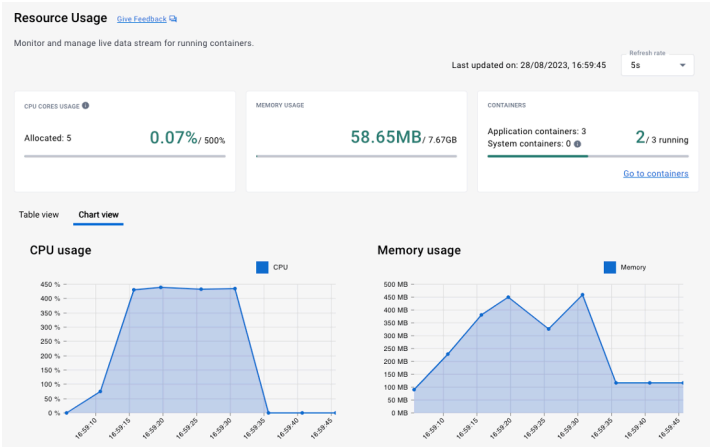
Gin aplikacija se pokreće u *Docker* kontejneru na portu 8082. Početak testiranja je u 11:01 sati. Korištenje resursa procesora i memorije je vidljivo na slici 4.29.. Najveće korištenje resursa procesora *Gin* aplikacije u petom testnom scenariju je oko 440%. Aplikacija je prilikom testiranja najviše koristila oko 550 MB, dok je u stanju mirovanja koristila oko 100 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 450 MB.



Slika 4.29. *Iskoristivost resursa prilikom testiranja petog testnog scenarija web okvira Gin*

4.6.4. Peti testni scenarij web okvira *Echo*

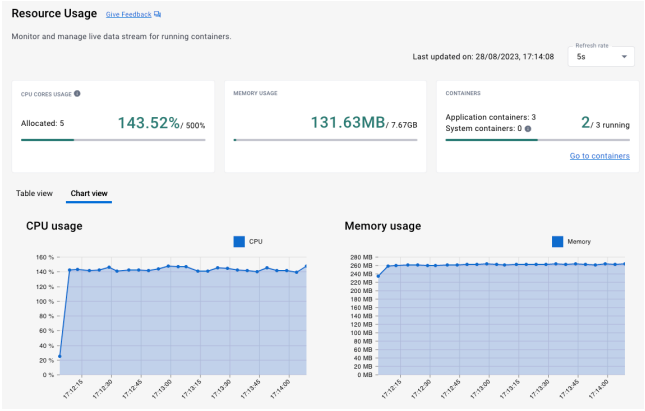
Echo aplikacija se pokreće u *Docker* kontejneru na portu 8083. Početak testiranja je u 16:59 sati. Korištenje resursa procesora i memorije je vidljivo na slici 4.30.. Najveće korištenje resursa procesora *Echo* aplikacije u petom testnom scenariju je oko 445% što je i najviše u usporedbi s ostalim aplikacijama prilikom testiranja petog testnog scenarija. Aplikacija je prilikom testiranja najviše koristila oko 450 MB, dok je u stanju mirovanja koristila oko 100 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 350 MB.



Slika 4.30. *Iskoristivost resursa prilikom testiranja petog testnog scenarija web okvira Echo*

4.6.5. Peti testni scenarij web okvira FastAPI

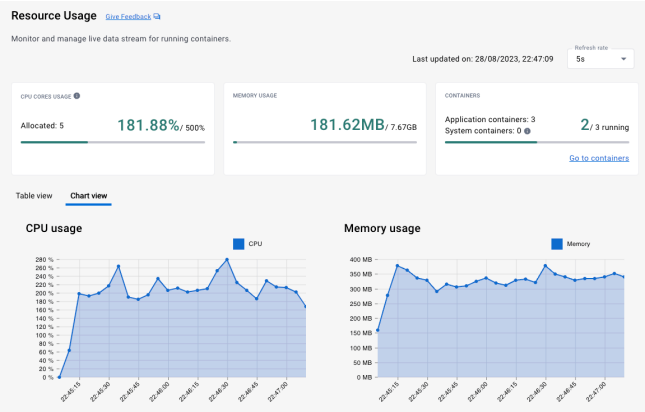
FastAPI aplikacija se pokreće u *Docker* kontejneru na portu 8084. Početak testiranja je u 17:12 sati. Korištenje resursa procesora i memorije je vidljivo na slici 4.31.. Najveće korištenje resursa procesora *FastAPI* aplikacije u petom testnom scenariju je oko 140% što je i najmanje u usporedbi s ostalim aplikacijama prilikom testiranja petog testnog scenarija. Aplikacija je prilikom testiranja koristila oko 260 MB, dok je u stanju mirovanja koristila oko 220 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 40 MB što je najmanje u usporedbi s ostalim aplikacijama prilikom testiranja petog testnog scenarija.



Slika 4.31. *Iskoristivost resursa prilikom testiranja petog testnog scenarija web okvira FastAPI*

4.6.6. Peti testni scenarij web okvira Flask

Flask aplikacija se pokreće u *Docker* kontejneru na portu 8085. Početak testiranja je u 22:45 sati. Korištenje resursa procesora i memorije je vidljivo na slici 4.32.. Najveće korištenje resursa procesora *Flask* aplikacije u petom testnom scenariju je oko 280%. Aplikacija je prilikom testiranja najviše koristila oko 390 MB, dok je u stanju mirovanja koristila oko 150 MB memorije. Razlika korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja je 240 MB.



Slika 4.32. *Iskoristivost resursa prilikom testiranja petog testnog scenarija web okvira Flask*

4.6.7. Usporedba rezultata petog testnog scenarija

U petom testnom scenariju *FastAPI* koristi najmanje resursa procesora, 140% od ukupno 500%, dok *Golang* web okviri koriste najviše, oko 440%. *FastAPI* koristi najmanje resursa memorije, u stanju mirovanja oko 220 MB dok u fazi testiranja koristi 40 MB više, ukupno oko 260 MB. *Spring Boot* koristi najviše resursa memorije, u stanju mirovanja 700 MB, dok u fazi testiranja koristi 150 MB više, ukupnu oko 850 MB. Najmanju razliku korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja imaju *FastAPI*, 40 MB. Najveću razliku korištenja resursa memorije aplikacije prilikom testiranja i u stanju mirovanja ima *Quarkus,* 400 MB kroz duži vremenski period.

Najkraće prosječno vrijeme izvršavanja pojedinačnog upita imaju *Java* web okviri; *Spring Boot* 173 milisekunde i *Quarkus* 107 milisekundi. Najduže prosječno vrijeme izvršavanja imaju *Python* web okviri; *FastAPI* 2638 milisekundi i *Flask* 3269 milisekundi. Median vrijednost izvršavanja upita je najmanja kod *Quarkus* web okvira, 77 milisekundi, a prati ga *Spring Boot* web okvir sa 138 milisekundi. Najveća medijan vrijednost izvršavanja upita je kod *Flask* web okvira, 5016 milisekundi. Najmanji postotak neizvršenih upita po testiranju ima *FastAPI* sa 0%, i iza njega su *Spring Boot* sa 1.06% i *Quarkus* sa 1.09% dok najveći imaju *Golang* web okviri, *Gin* 8.61% i *Echo* 7.28%. Najmanje operacija po sekundi ima *Flask*, svega 75.9 operacija po sekundi, dok najviše operacija po sekundi imaju *Java* web okviri; *Spring Boot* 1528.9 operacija po sekundi i *Quarkus* 2629.3 operacija po sekundi. Najmanja prosječna veličina odgovora na upit je kod *Quarkus*-a, 187.5 bita, a najveća kod *Flask-a*, 283.1 bita. Ukupno trajanje izvršavanja testa je najsporije kod *Flask* web okvira, 3 minute i 18 sekunde, dok je najbrže kod *Java* web okvira, za *Spring Boot* ono iznosi 9 sekundi i za *Quarkus* iznosi 5 sekundi (tablica 4.5.).

Tablica 4.5. *Metrički podatci testiranja performansi petog testnog slučaja*

Web okvir	**Broj uzoraka**	**Prosjek / milisekunda**	**Median / milisekunda**	**Greška / %**	**Propusnost / operacija/sekunda**	**Prosjek / bit**	**Trajanje / minuta:sekunda**
Spring Boot	15000	173	138	1.06	1528.9/sec	238.4	0:09
Quarkus	15000	107	77	1.09	2629.3/sec	187.5	0:05
Gin	15000	442	444	8.61	623.9/sec	229.7	0:24
Echo	15000	428	447	7.28	642.2/sec	239	0:23
FastAPI	15000	2638	2650	0	112.5/sec	217.3	2:13
Flask	15000	3269	5016	2.09	75.9/sec	283.1	3:18

4.7. Veličine aplikativnih *Docker* slika i aplikacija

Kod korištenih radnih okvira u ovom radu, najmanje su aplikacije *Golang* programskog jezika sa izuzetkom *Quarkus*-a koji ima najmanju aplikaciju od svega 22 KB. Najveće su aplikacije *Python* programskog jezika, one nisu kompilirane i sadrže u sebi virtualne okoline bez kojih ne bi mogle biti pokrenute. Najveća je aplikacija radnog okvira *FastAPI* i njezina veličina iznosi 74 MB (slika 4.6.).

Tablica 4.6. *Veličine aplikacija*

Web okvir	**Veličina aplikacije**
Spring Boot	42.3 MB
Quarkus	22 KB
Gin	21.5 MB
Echo	18.7 MB
FastAPI	74 MB
Flask	37 MB

Kod korištenih radnih okvira u ovom radu, najmanje su aplikativne *Docker* slike *Python* programskog jezika; *FastAPI* je veličine 269.13 MB i *Flask* je 231.95 MB, dok su najveće aplikativne *Docker* slike *Golang* programskog jezika, *Gin* je veličine 1.19 GB i *Echo* je 1.14 GB.

Tablica 4.7. *Veličine aplikativnih Docker slika*

Web okvir	**Veličina *Docker* slika**
Spring Boot	537.94 MB
Quarkus	453.28 MB
Gin	1.19 GB
Echo	1.14 GB
FastAPI	269.13 MB
Flask	231.95 MB

ZAKLJUČAK

U ovom diplomskom radu testirane su performanse modernih radnih okvira za kreiranje web usluga. Bolje performanse imaju radni okviri kompiliranih programskih jezika, dok se *Python* servisi mogu koristiti za razvijanje web usluga za rad s velikom količinom podataka zbog ekosistema biblioteka koji se i dalje razvijaju u okvirima visoko obrazovnih ustanova. *Quarkus* ima najmanju veličinu aplikacije što je dobro za razvoj aplikacije u stvarnom vremenu za računalstvo u oblaku, jer se takve aplikacije mogu brzo podignuti i ne zauzimaju veliki prostor, što znači monetarno bolje upravljanje resursima. *GraalVM* kompilira aplikaciju i uklanja sve nepotrebne dijelove biblioteka koje aplikacija ne koristi, a time aplikacija dobiva bolje performanse i manji memorijski otisak, no znatno gubi na performansama pri izvršenju zahtjevnijih matematičkih zadataka, kao u trećem testnom scenariju. *Spring Boot* ima dobra prosječna vremena, odmah iza *Golang* radnih okvira, i najbolja median vremena izvršavanja upita, osim u petom testnom slučaju, nizak postotak neizvršenih upita po testiranju, brzo obavlja teže matematičke zadatke, odličan je i brz u izvršavanju *SQL* upita na bazu, no isto tako *ORM* je jako efikasan i neznatno lošiji u brzini u odnosu na izvršavanje običnih *SQL* upita. Treba napomenuti da *Java* koristi *JVM* koji periodično čisti memoriju, no time i koristi resurse procesora na radnje koje povećavaju vrijeme izvršavanja upita. Veličina datoteke je osrednja, manja od *Python* aplikacija kojima je potrebno postaviti virtualno okruženje kako bi se mogle izvršavati što zahtjeva više memorijskog prostora, no veća od *Quarkus* i radnih okvira *Golang* programskog jezika. Trajanje izvršavanja testnih slučajeva aplikacije pisane u *Spring Boot* radnom okviru je jedno od najboljih, po brzini uvijek na prvom ili drugom mjestu. Radni okviri pisani u *Java* programskom jeziku koriste najviše memorijskih resursa u svim testnim slučajevima. Radni okviri *Golang*-a su vrlo dobri, imaju relativno malu veličinu aplikacija, koriste jako malo resursa memorije u čemu *Echo* prednjači, relativno nizak postotak neizvršenih upita po testiranju i najbolje vrijeme izvršavanja u prva tri testna scenarija i lošije vrijeme izvršavanja u slučajevima povezanim s bazom podataka. Također, koriste maksimalan broj resursa procesora jer imaju dobro razvijene istodobne funkcije izvršavanja na više procesora odjednom. Dobro rade pod opterećenjem težih matematičkih kalkulacija kao i aplikacije pisane u *Spring Boot-u*. Korištenje resursa procesora raste s optimalno raste s težinom zadatka koji izvršavaju. Ono što je loše kod *Golang* aplikacija je ne toliko efikasan *ORM* i rad s bazom s standardnom bibliotekom, gdje imaju lošija vremena izvršavanja kao i dosta veliki postotak neizvršenih upita po tesitranju. *Golang* aplikacije je moguće aplikativno bolje optimizirati u radu sa standardnom bibliotekom za rad s bazom podataka. Oba radna okvira Golang programskog jezika su slična po metričkim rezultatima, vrlo optimalno

koriste resurse procesora i memorije. Veličine *Docker* slika se mogu bolje optimizirati korištenjem drugih slika na koje se mogu instalirati samo potrebni programi, što bi se moglo napraviti u nekom budućem testiranju. Ako treba izabrati programski jezik za razvijanje mikro usluga, po podatcima testiranja treba uzeti jedan od *Java* ili *Golang* radnih okvira. *Quarkus* se ne bi trebao koristiti za aplikacije koje izvršavaju teže računske operacije. *Python* ima lošije performanse od svih testiranih radnih okvira no može se koristiti kao izdvojena mikro usluga za rad s velikom količinom i strojnom učenju zbog postojećeg ekosistema koji ga razvija u tom smjeru. Pri odabiru radnog okvira treba uzeti u obzir performanse samog okvira, područje u kojem okvir i programski jezik posjeduje zrele biblioteke, kao i sam ekosistem zajednice koja koristi radni okvir i razvija ga, zbog problema na koje možemo naići prilikom razvijanja aplikacija i odgovora koje možemo pronaći za isti. Ne možemo pogriješiti s odabirom ijednog od navedenih okvira, no za bolje performanse potrebno je napraviti detaljniju analizu domene problema koju sama aplikacija rješava te uzeti u obzir i resurse s kojim raspolažemo. U daljnjem testiranju trebalo bi se napraviti i testiranje ponašanja predmemorije pojedinog radnog okvira i testirati aplikacije na nekom univerzalnom poslužitelju u računalstvu u oblaku.

LITERATURA

[1] S. Lewis, web services, https://www.techtarget.com/searchapparchitecture/definition/Web-services [29. lipanj 2023.]

[2] T. Cleo, What Are Web Services? Easy-to-Learn Concepts with Examples, https://www.cleo.com/blog/knowledge-base-web-services [29. lipanj. 2023.]

[3] A. Monus, SOAP vs REST vs *JSON* – a 2023 comparison, https://raygun.com/blog/soap-vs-rest-vs-*JSON*/ [29. lipanj 2023.]

[4] Q. H. Mahmoud, Service – Oriented Arhitecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI), https://www.oracle.com/technical-resources/articles/*Java*se/soa.html [29. lipanj 2023.]

[5] J. Olawanle, What is a Framework? Software Frameworks Definition, https://www.freecodecamp.org/news/what-is-a-framework-software-frameworks-definition/ [29. lipanj 2023.]

[6] R. K. Sharma, What is a Framework in Programming? And Why You should Use One, https://www.netsolutions.com/insights/what-is-a-framework-in-programming/ [29. lipanj 2023.]

[7] A. S. Gillis, performance testing, https://www.techtarget.com/searchsoftwarequality/definition/performance-testing [29. lipanj 2023.]

[8] T. Simond, Everything you need to know about load testing, https://queue-it.com/blog/load-testing/ [29. lipanj 2023.]

[9] N. Cohen, What is Performance Testing vs. Load Testing vs. Stress Testing?, https://www.blazemeter.com/blog/performance-testing-vs-load-testing-vs-stress-testing [29. lipanj 2023.]

[10] M. Hixson, M. Smith, N. Brady, Welcome to TechEmpower Framework Benchmarks (TFB), https://github.com/TechEmpower/FrameworkBenchmarks [30. lipanj 2023.]

[11] M. Rabbaa, Web Frameworks Benchmark, https://github.com/the-benchmarker/website [30. lipanj 2023.]

[12] Tech Stack Intelligence, https://stackshare.io/ [25. kolovoz 2023.]

[13] G. Shubhangi, Introduction to *Java*, https://www.geeksforgeeks.org/introduction-to-*Java*/ [30. lipanj 2023.]

[14] R. Gerardo wtih group of authors named as Baeldung, Learn *Spring Boot*, https://www.baeldung.com/spring-boot [30. lipanj 2023.]

[15] J. Thorben, Design Patterns Explained – Dependency Injection with Code Examples, https://stackify.com/dependency-injection/ [30. lipanj 2023.]

[16] F. Hasan, What is Inversion of Control?, https://www.educative.io/answers/what-is-inversion-of-control [30. lipanj 2023.]

[17] C. Duffy, Unveiling the Top 5 *Java* Frameworks for Building Microservices: How to Choose the Best Framework for Your Project, https://levelup.gitconnected.com/unveiling-the-top-5-*Java*-frameworks-for-building-microservices-how-to-choose-the-best-framework-57401832641f [30. lipanj 2023.]

[18] R. Pike, Go at Google: Language Design in the Service of Software En*Gin*eering, https://go.dev/talks/2012/splash.article [30. lipanj 2023.]

[19] H. Shafqat, What is a *Gin* Web Framework in *Golang*, https://linuxhint.com/*Gin*-web-framework-*Golang*/ [30. lipanj 2023.]

[20] S. Das, Building Microservice using *Golang Echo* framework, https://medium.com/cuddle-ai/building-microservice-using-*Golang*-*Echo*-framework-ff10ba06d508 [30. lipanj 2023.]

[21] S. Worsley, What is *Python*? – The Most Verstaile Programming Language, https://www.datacamp.com/blog/all-about-*python*-the-most-versatile-programming-language [30. lipanj 2023.]

[22] F. Malik, Build And Host Fast Dana Science Applications Using *FastAPI*, https://towardsdatascience.com/build-and-host-fast-data-science-applications-using-*FastAPI*-823be8a1d6a0 [30. lipanj 2023.]

[23] M. Makai, Full Stack *Python*, https://www.fullstack*python*.com/*Flask*.html [30. lipanj 2023.]

[24] A. Mittal, (Rest API using Spring Boot) Part 2 Adding Model, Service, Controller, and Dao Implementation, https://medium.com/javarevisited/rest-api-using-spring-boot-part-2-adding-model-service-controller-and-dao-implementation-697284b4ff38 [30. lipanj 2023.]

[25] C. Venti, What is *Docker*, https://opensource.com/resources/what-*Docker* [30. lipanj 2023.]

[26] B. Arunachalam, What is *Docker* Compose? How to Use it with an Example, https://www.freecodecamp.org/news/what-is-*Docker*-compose-how-to-use-it/ [30. lipanj 2023.]

[27] I. Gaba, JMeter Load Testing: A Comprehensive Guide, https://www.simplilearn.com/tutorials/jmeter-tutorial/jmeter-load-testing [30. lipanj 2023.]

[28] K. K Singh, Understand and Analyse Aggregate Report in Jmeter, https://www.oodlestechnologies.com/dev-blog/understand-and-analyse-aggregate-report-in-jmeter/ [30. lipanj 2023.]

SAŽETAK

U ovom diplomskom radu razvijeno je i uspoređeno šest aplikacija koji imaju isti *API* kojim su testirana ponašanja aplikacije u pet testnih slučajeva. Korištena su po dva radna okvira u tri različita programska jezika, Javi, Golangu i Pythonu. Aplikacije su sastavljene u tri sloja; sloj upravljača, servisni sloj i sloj pristupa podatcima. Testirani slučajevi su JSON serijalizacija i deserijalizacija, obavljanje teže matematičke operacije na servisnom sloju aplikacije bez rada s bazom podataka, komunikacija s bazom podataka preko pisanih SQL upita i komunikacija s bazom podataka preko ORM-a. Aplikacije su testirane na računalu te su korišteni *Docker i Docker Compose* za podizanje aplikacija i baze podataka u kontejnerima. Cilj je usporediti metričke rezultate vremena izvršavanja upita, postotka greške, propusnosti, prosječne veličine odgovora na upit, ukupno trajanje izvršavanja svih upita te korištenje resursa procesora i memorije pri izvršavanju testiranja performansi u različitim testnim slučajevima. Rezultati testiranja pokazali su kako je najbolje koristiti *Spring Boot* radni okvir jer ima ponajbolje rezultate u svim slučajevima ako se raspolaže sa dovoljno memorijskih resursa. *Quarkus* radni okvir se može koristiti za razvoj aplikacija u računalnom oblaku jer ima vrlo malu veličinu aplikacije i dobre rezultate, osim pri izvršavanju težih matematičkih operacija, te kao i *Spring Boot* koristi puno resursa memorije zbog virtualne mašine koju koristi. *Golang* aplikacije su odlične za razvoj aplikacija koje se integriraju sa drugim sustavima i istodobno trebaju prikupljati informacije s različitih sustava u realnom vremenu, te ne rade pretjerano puno s bazama podataka, jer imaju loše metričke rezultate u četvrtom i petom testnom scenariju. Metrički rezultati *Python* radnih okvira su lošiji od ostalih no odlični su za razvoj mikro usluga koje rade sa velikom količinom podataka i na strojnom učenju zbog biblioteka koje posjeduje.

Ključne riječi: *baza podataka, radni okvir, REST, testiranje performansi, web usluge*

ABSTRACT

In this master's thesis, six applications were developed and compared, all of which utilize the same API for testing application behaviors across five test cases. Two frameworks were used in three different programming languages: *Java*, *Golang*, and *Python*. The applications are structured into three layers: the controller layer, the service layer, and the data access layer. The tested cases include JSON serialization and deserialization, performing complex mathematical operations on the service layer without database interaction, database communication through handwritten SQL queries, and database communication through an ORM. The applications were tested on a computer using *Docker* and *Docker Compose* to deploy them along with databases in containers. The goal was to compare metric results such as query execution time, error percentage, throughput, average response size, total execution time for all queries, and CPU and memory resource utilization during performance testing across different test cases. The test results indicated that the *Spring Boot* framework performed the best in all cases when sufficient memory resources were available. *Quarkus*, while suitable for cloud-based application development due to its small application size and good results, struggled with more complex mathematical operations and consumed significant memory resources, similar to *Spring Boot*, due to the virtual machine it employs. *Golang* applications proved excellent for systems integration and real-time data gathering from various sources but had poorer metrics in the fourth and fifth test scenarios involving heavy database usage. *Python* frameworks exhibited inferior metrics compared to others but are well-suited for developing microservices dealing with large data volumes and machine learning tasks due to their extensive libraries.

Keywords: *database, framework, REST, performance testing, web services*

ŽIVOTOPIS

Ivan Lončar rođen je 21.11.1989. u Osijeku, Hrvatska. Godine 2003. upisuje Prirodoslovno-matematičku gimnaziju u Osijeku. U srednjoj školi pokazuje poseban interes za fiziku, astronomiju, matematiku i informatiku. Godine 2007. upisuje preddiplomski sveučilišni studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija. Temom završnog rada "Izrada animacija za interaktivna predavanja u programskom okružju FLASH" stječe akademski naziv sveučilišnog prvostupnika inžinjera računarstva. Nakon toga upisuje diplomski sveučilišni studij procesnog računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija. Bio je član Studenskog zbora i Studentskog vijeća fakulteta. Bio je na dvije stučne prakse preko studentske volonterske udruge IAESTE čiji je bio dugogodišnji član. Trenutno radi u International Value Services kao programer.

_____

Potpis autora