

*Prolećni semestar, 2023/24*

*PREDMET: CS230 – Distribuirani Sistemi*

Projektni Zadatak: Distribuirana Chat Aplikacija

Ime i prezime: **Petar Otovic 5460**

Broj indeksa: **5460**

Datum izrade: **26-5-2024**

Profesor: **Nemanja Zdravkovic**

Asistent: **Andjela Grujic**

**Sadrzaj**

Contents

[1. Uvod u projekat 3](#_Toc167652356)

[2. Arhitektura 4](#_Toc167652357)

[3. Pokretanje 13](#_Toc167652358)

# Uvod u projekat

Cilj ovog projekta je razvoj distribuirane chat aplikacije koja omogućava korisnicima da razmenjuju poruke preko vise socketa. Projekt je realizovan kao deo završnog rada iz predmeta CS230 - Distribuirani Sistemi. Aplikacija je dizajnirana tako da demonstrira ključne koncepte distribuiranih sistema, kao što su skalabilnost, raspodela opterećenja i efikasno upravljanje konekcijama.

**Pregled aplikacije**

Aplikacija se sastoji od tri glavne komponente:

* **Chat Aplikacija (Klijent):** Frontend deo sistema u vidu mobilne aplikacije koji korisnicima omogucava da se prijave, povezu sa serverom i komuniciraju sa drugim korisnicima. Aplikacija je razvijena koristeci Flutter.
* **Load Balancer:** Ova komponenta sluzi da rasporedi korisnike na najmanje optereceni server, ako ih ima vise. Load balancer zna u svakom trenutku koliko koriniska ima na svakom serveru i preusmerava korisnike na one servera gde ima najmanji broj konekcija, cime se postize raspodela resusrsa i bolje performanse.
* **Server:** Backend komponenta koja se povezuje sa klijentima i omogućava razmenu poruka. Server upravlja korisničkim sesijama, skladišti poruke i omogućava real-time komunikaciju između korisnika. Serveri su dizajnirani tako da mogu raditi u paraleli, omogućavajući horizontalnu skalabilnost sistema.

**Funkcionalnosti**

* Prijava Korisnika: Korisnici mogu da se prijave sa izabranim kornisickim imenom na server.
* Real-time Komunikacija: Korisnici mogu da razmenjuju poruke sa drugim korisnicima u realnom vremenu preko socket-a.
* Raspodela Opterećenja: Load balancer dinamcki usmerava korisnike na najmanje opterecene servere.
* Skalabilnost: Sistem je dizajniran da podrzi dodavanje novih servera radi povecanja kapaciteta i pouzdanosti.

**Tehnologije**

* Flutter: Koriscen za izradu forntend dela aplikacije u vidu mobilne aplikacije.
* Kotlin: Koriscen za izradu Load Balancer-a i servera chat aplikacije.

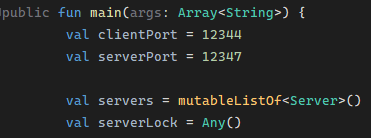
# Arhitektura

* **Load Balancer**

Load Balancer je deo koji ostvaruje funkcionlanost ove apliakcije. Klijent se ne moze povezati na server bez load blanacera, zato sto klijent prvo salje zahtev da bude rasporedjen an server pre nego sto se poveze na isti. Load balancer vraca informacije za konekciju kao sto su ip i port na kome je pokrenut server, koji ce klijent iskoristiti kako bi se povezao.

Load balancer slusa na dva socketa, svaki na svom thred-u. Jedan socket sluzi za osluskivanje konekcija od strane klijenata, dok drugi socket sluzi za osluskivanje konekcija od strane servera, koji salju zahtev kako bi se javili da su dostupni za konekciju.

Po default-u dva socketa su 12344 za klijente i 12347 za servere. Moze se desiti da pri pokretanju dodje do greske zato sto je jedan ili zato sto su oba socketa zauzeta, pa je potrebno promeniti ih.



Slika 1: Sokceti za server i klijent konekcije

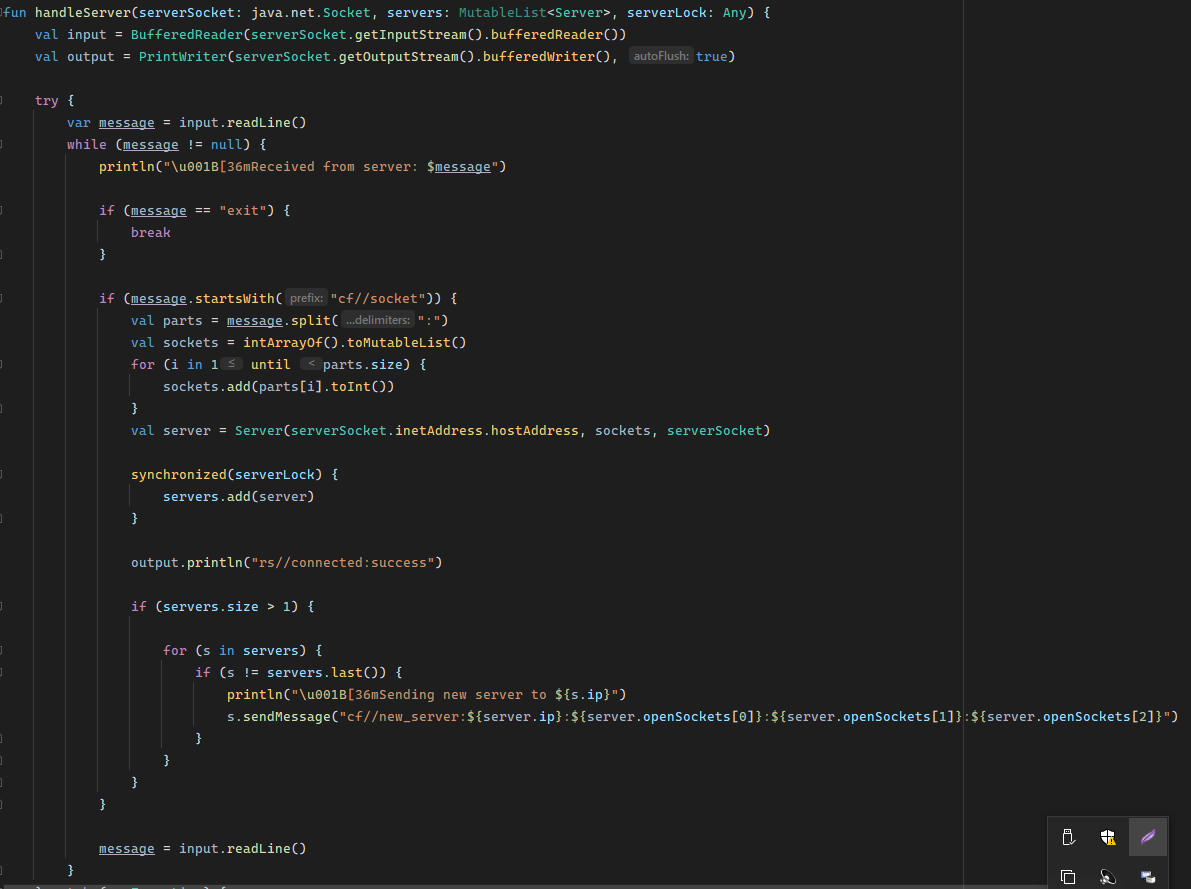
Lista **servers** ce cuvati sve dostupne servere koji se budu registrovali. **serverLock** sluzi kako bi se sprecio conccurent modification excpetion prilikom modifikacije liste servera u thread-ovima.

Load balancer pokrece 2 thread-a od kojih je jedan za obradu zahteva od servera



Slika 2: Thread za obradu konekcija od strane servera

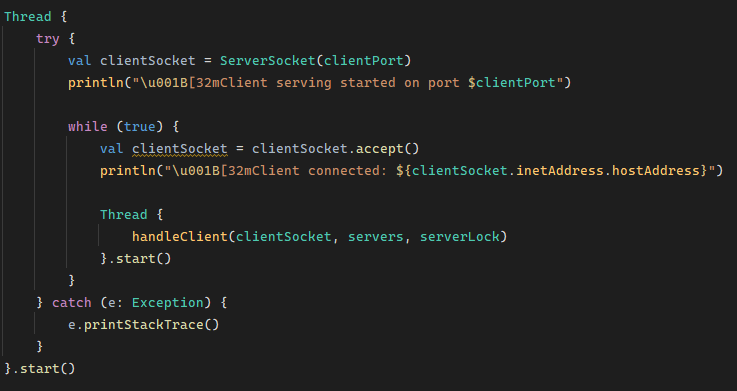
Kada load balancer dobije zahtev za kreiranje servera poziva se metoda **handleServer** koja ce obraditi zahtev i smestiti server u listu servera.



Slika 3: Metoda za obradu zahteva od servera i smestanje u listu

Metoda **handleServer** pokrece beskonacnu petlju i obradjuje sve poruke koje stignu od strane servera. Server ce u pocetku poslati svoje konfiguracione podatke tj. Sockete na kojima je pokrenut, dok nam je ip adresa servera dostupna preko socketa na koji je ostvarena konekcija na server.

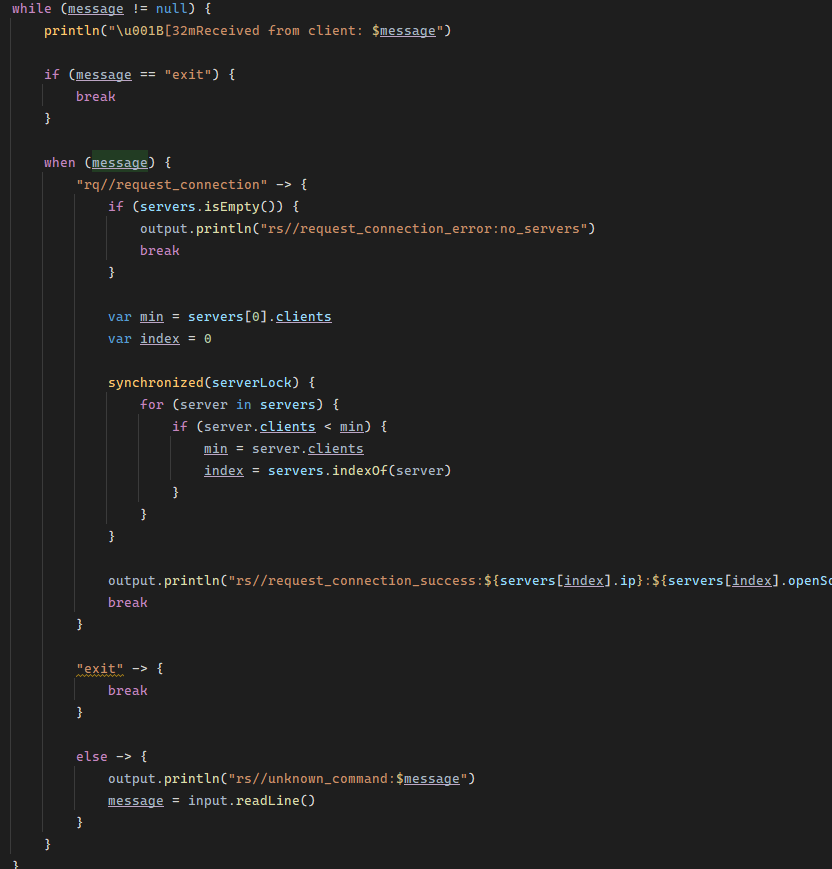
Drugi thread koji se pokrece ce osluskivati za konekcije koje su poslate od strane klijenta.



Slika 4: Thread za osluskivanje konekcija od strane klijenta

Kada zahtev od strane klijenta bude primljen, poziva se metoda **handleClient** koja ce obraditi zahtev.

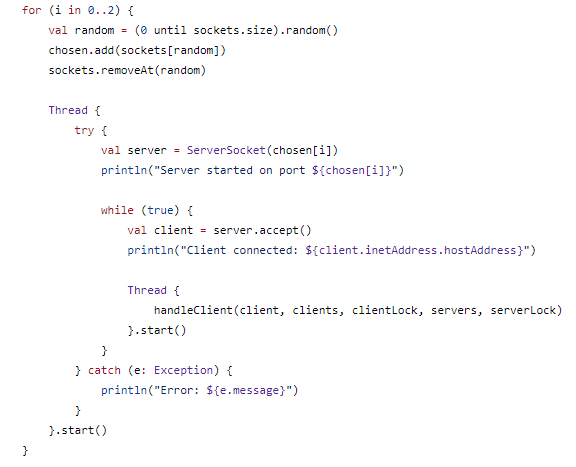
Metoda **handleClient** ce proveriti da li ima dostupnih servera i poslati klijentu informacije o serveru u slucaju da je dostupan.



Kada load balancer posalje podatke o serveru on zatvara konekciju sa klijentom, ne drzi je otvorenom.

* **Server**

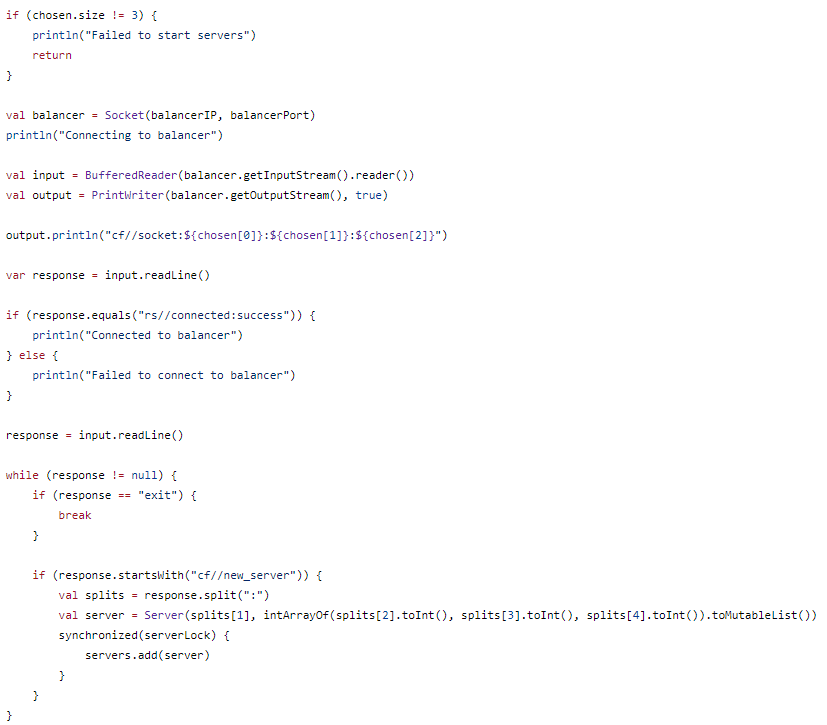
Server komponenta sluzi kako bi povezala klijente i omogucila im da komuniciraju izmedju sebe. Svaki server se pokrece na 3 socket i na njima ce slusati za nadolazece konekcije od strane klijenta.



Slika 5: Pokretanje servera na 3 nasumicna socketa

Iz niza od 6 port-ova se biraju nasumicno 3 na kojima ce biti pokrenut server a na kojima ce se povezivati klijenti. Svrha ovoga je da omoguci glavnu funkcionalnost ovog projekta, tj. Komunikacija klijenta preko vise socketa.

Nakon sto se server pokrene, salje svoje konfiguracione podatke **load balanceru**, kako bi postao registrovan i kako bi klijenti mogli da se povezu na njega.



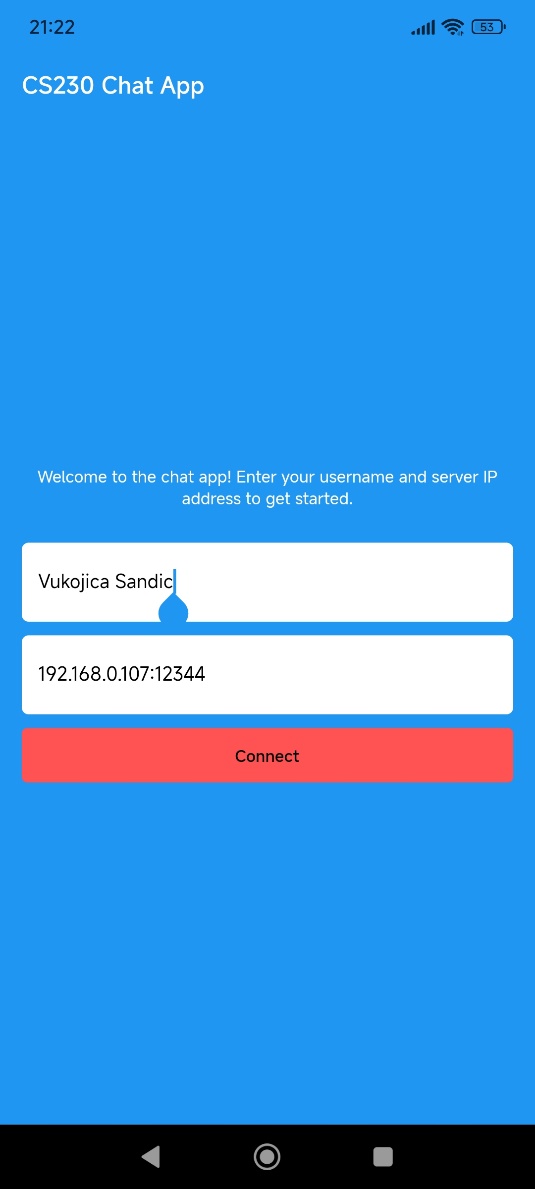
Slika 6: slanje konfiguracionih podataka load balanceru

Kada se ostvari konekcija sa klijentom pokrece se metoda **handleClient.** U ovoj metodi se obradjuju svi zahtevi od strane klijenta.

Moze se desiti da su korisnici na skroz razlicitim serverima (ne socketima, serverima). U ovom slucaju kada komuniciraju, korisnik koji salje poruku je salje na server. Korisnik koji prima poruku se prvo trazi na trenutnom serveru, ako nije pronadjen, server salje ostalim serverima poruku koju je poslao prvi korisnik. Svi ostali serveri traze tog korisnika kod sebe i ako ga pronadju, dostavljaju mu poruku. Ovaj nacin komunikacije (broadcast) je neefektivan i zauzima previse resursa, pogotovu kada ima mnogo servera. Jedan nacin da se ovo resi je da se povezani korisnici cuvaju u bazi sa njihovom server lokacijom. Server bi kada trazi drugog korisnika, samo procitao njegovu lokaciju u bazi i poslao tom serveru poruku, bez slanja svakom serveru pojedinacno.

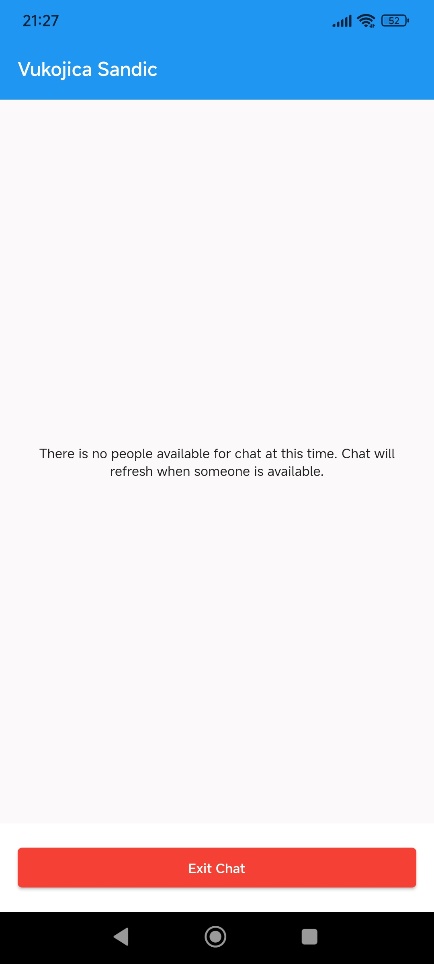
* **Klijent Deo**

Klijent deo je, kao sto je vec receno, razvijen u Flutter radnom okviru za izradu intuitivnih korisnicikih interfejsa. Sastoji se od 3 ekrana ukupno. Na prvom ekranu se unose korisnicko ime, koje ce se prikazivati ostalim korisnicima i adresu i port load balancera na koji treba da se poveze korisnik.



Slika 7: Ekran za registraciju korisnika

Drugi ekran, lobby ekran, je mesto gde se prikazuju svi korisnici dostupni za chat.



Slika 8: Lobby ekran, nijedan korisnik nije dostupan

Postoji i opcija za izlazak, tj vracanje nazad na pocetni ekran.



Slika 9: Lobby ekran, sa dostupnim korisnicima

Klikom na korisnika otvara se ekran za razmenjivanje poruka.



Slika 10: Ekran za razmenjivanje poruka

# Pokretanje

Prvo je potrebno pokrenuti **Load Balancer.** U slucaju da su default-ni socketi zauzeti neophodno je da se promene. Nakon toga potrebno je otkriti ip adresu load balancer-a. Otvara se **cmd,** kuca se komanda ipconfig i uzima se Ipv4 adresa. Nakon toga pokrece se bar jedan server. Neophodno je pormeniti pormenljivu **balancerIP** na pocetku main funckije u serveru sa IP adresom koju ste uzeli iz cmd-a. Nakon toga se pokrece server, koji ce se automatski povezati na loda balancer. U slucaju uspesne konekcije na load balancer prikazace se poruka u konzoli kao i na load balanceru.

Za pokretanje klijenta je potrebno instalirati Dart i Flutter SDK. Potrebna su dva fizicka uredjaja (telefona) ili emulator. Kada se instalira potreban SDK, projekat se pokrece komandom **flutter run** koja ce pokrenuti aplikaciju na telefonu. Pokrenuti aplikaciju na oba telefona. Treba uneti **ip adresu** koju ste prethodno uzeli iz cmd-a i uneti je u donjem polju, kao i socket na kojem load balancer slusa za klijentske konekcije. Nakon toga se mozete ulogovati i koristiti chat aplikaciju.