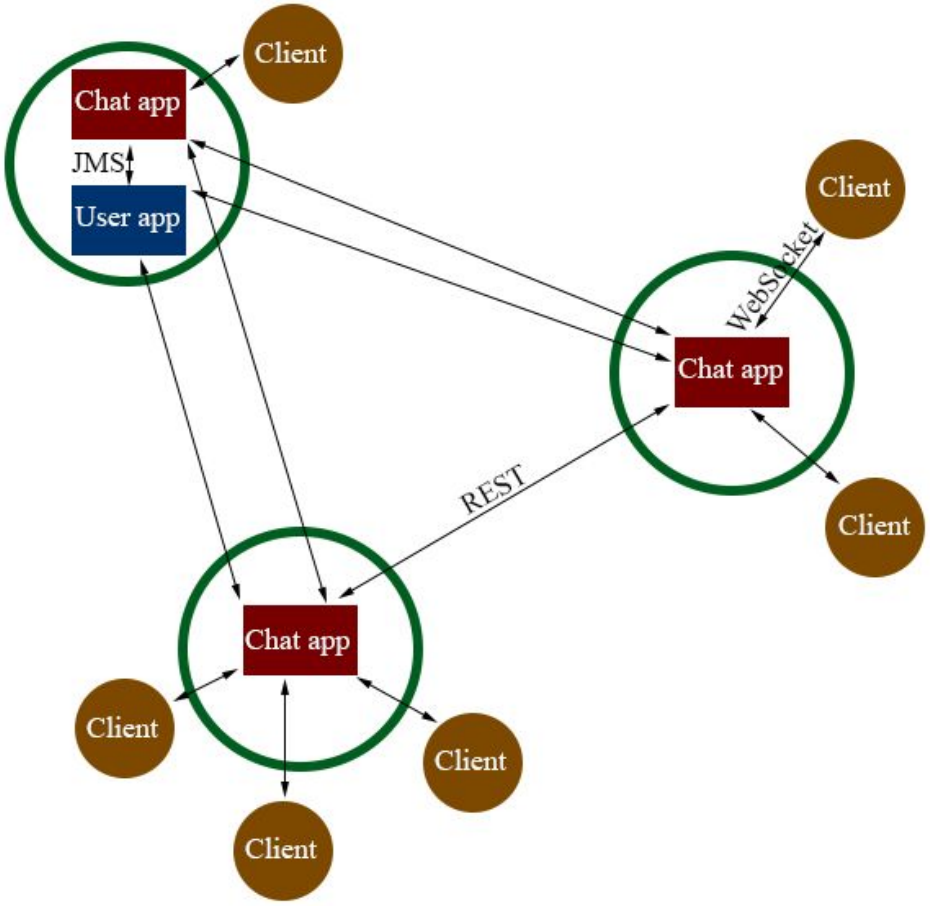
ČET APLIKACIJA – PRVA FAZA

Potrebno je napraviti dve EJB aplikacije koje će raditi na WildFly aplikativnom serveru.

Prva aplikacija **UserApp** će raditi na jednom čvoru i njen zadatak će biti da upravlja korisnicima. Ona treba da omogući **registraciju, prijavu na sistem i odjavu korisnika, dodavanje prijatelja, pretragu korisnika i rad sa grupama**. Korisnici sa njom interaguju isključivo indirektno, putem aplikacije za čet.

Druga aplikacija **ChatApp** predstavlja web aplikaciju za čet i ona će biti postavljena na više čvorova. Svu interakciju sa sistemom korisnik vrši putem ove aplikacije. Server-side aplikacija ima tri grupe funkcionalnosti i to **upravljanje klasterom ChaApp aplikacija, delegiranje akcije vezane za upravljanje korisnicima UserApp-u i obrada čet poruka**. Klijentska aplikacija je Angular ili React aplikacija koja korisniku treba da prikaže stranice za registraciju korisnika, prijavu na sistem, profil korisnika i čet.

Arhitektura sistema, uz protokole za komunikaciju između različitih aktera je prikazana na slici 1. Za **master čvor** ćemo smatrati čvor na kom su smeštene **ChatApp** i **UserApp**.



Slika 1. Arhitektura sistema

MODEL PODATAKA

Određeni podaci se trebaju trajno čuvati u bazi podataka (preporuka je da to bude NoSQL baza npr. MongoDB). U bazi se trebaju čuvati podaci o korisnicima, grupama i razmenjenim porukama. Bazu podataka pokrenuti na master čvoru (*ukoliko želite možete obezbediti decentralizovano čuvanje podataka*). Obe aplikacije treba da koriste bazu, UserApp za rad sa korisnicima, a ChatApp za čuvanje poruka i pristup grupama kako bi se ustanovilo koji korisnici pripadaju određenoj grupi.

Korisnici su opisani sledećim podacima: **korisničko ime**, **lozinka**, **ime**, **prezime i host** na kom je on trenutno ulogovan. Podatak o hostu se ne treba čuvati u bazi pošto se on dinamički dodeljuje korisniku kada se on prijavi na sistem (host putem kog se prijavio). Korisnici mogu kreirati prijateljstva i to se treba trajno čuvati. Takođe, korisnici mogu kreirati grupe i dodavati druge korisnike u njih.

Gupa je opisana sledećim podacima: **id, ime i članovi (korisnici)**. Grupe postoje kako bismo omogućili čet između više korisnika.

Poruka je opisana sledećim podacima: **id**, **pošiljalac**, **primalac**, **datum i vreme slanja, i sadržaj**. Voditi računa da kada se poruka šalje grupi imamo više primalaca.

Host je opisan sledećim podacima: **adresa i alias**. Adresa je IP adresa + port hosta, dok je alias zapravo naziv hosta putem koga ga možemo tražiti, kako ne bismo uvek morali da unosimo adresu. Podatke o hostu ne treba čuvati u bazi.

USER APP

UserApp aplikacija će biti postavljena na jedan čvor i treba da obezbedi sledeće funkcionalnosti:

- Registracija novih korisnika
- Prijava korisnika na sistema
- Odjava korisnika sa sistema
- Rad sa prijateljima
    - Dodavanje
    - Brisanje
- Rad sa grupama
    - Kreiranje
    - Brisanje
    - Dodavanje novog korisnika u grupu
    - Uklanjanje korisnika iz grupe - korisnik koji je kreirao grupu ima pravo da izbaci nekoga iz grupe. Takođe, korisnik može sam napustiti grupu.

ChatApp će sa UserApp komunicirati putem REST endpoint-a (u slučaju da se nalaze na različitim čvorovima), odnosno putem JMS-a (u slučaju da se nalaze na istom čvoru).

Lista aktivnih korisnika se čuva u ovoj i Chat aplikaciji. U ovoj se nalazi “glavna” lista i putem master Chat aplikacije će se o izmenama iste obaveštavati ostale Chat aplikacije (kasnije opisano). Lista aktivnih korisnika treba da bude uskladištena u RAM-u. Host polje korisnika treba da se čuva u „bazi“ aktivnih, odnosno prijavljenih korisnika i generiše se svaki put kada se korisnik prijavi na sistem.

CHAT APP – SERVER SIDE

ChatApp aplikacija će biti postavljena na više čvorova i treba da podrži tri grupe funkcionalnosti.

1. Funkcionalnosti vezane za upravljanje klasterom podrazumevaju sledeće:

- Registraciju čvora u klaster
- Uklanjanje čvora iz klastera

Svaki ne-master čvor u konfiguracionoj datoteci treba da čuva podatke o master čvoru. Kada se podigne ne-master čvor na adresu, pročitanu iz konfiguracione datoteke, šalje zahtev za registraciju i ukoliko je zahtev uspešan dobija listu svih prethodno registrovanih čvorova klastera, koju beleži kod sebe. Zatim šalje dodatni zahtev za dobijanje svih aktivnih korisnika UserApp-u master čvora i beleži dobijenu listu korisnika kod sebe. Master čvor, nakon što je uspešno registrovao novi čvor prolazi kroz listu čvorova i šalje register zahtev sa adresom i alijasom novog čvora svim ostalim čvorovima, kako bi oni zabeležili da se nov čvor dodao u sistem.

Sličan proces se dešava kada uklanjamo čvor iz klastera. Ne-master čvor šalje masteru zahtev za uklanjanje iz klastera, nakon čega master šalje svim ostalim čvorovima zahtev da uklone odgovarajući čvor.

2. Funkcionalnosti delegiranja akcija vezanih za upravljanje korisnicima će biti aktivirane od strane klijentske aplikacije putem WebSocket-a. Kada serverska strana ChatApp aplikacije prihvati poruku vezanu za rad sa korisnicima ona parsira poruku i pravi određen REST zahtev ka UserApp aplikaciji (ako nisu na istom čvoru) ili formira JMS poruku (ako jesu). Za sve funkcionalnosti koje nudi UserApp, ChatApp mora imati REST endpoint-e koji se ponašaju kao proxy ka endpoint-ima User app-a.

Osim toga, ChatApp treba da ponudi i sledeću funkcionalnost:

- Dodavanje korisnika u listu aktivnih
- Uklanjanje korisnika iz liste aktivnih

Kada se korisnik uspešno prijavi na sistem UserApp treba da, putem JMS-a kaže ChatApp aplikaciji master čvora da se prijavio novi korisnik. Zatim, ChatApp prolazi kroz listu svih čvorova i pravi zahtev za dodavanje novog aktivnog korisnika, kako bi i drugi čvorovi bili ažurirani. Zatim, putem WebSocket-a, se ažurira klijentska aplikacija i korisnik se prikazuje kao aktivan. Ista priča važi za odjavu sa sistema, gde se šalju zahtevi za uklanjanje korisnika.

3. Funkcionalnost obrade čet poruka podrazumeva da korisnik kroz klijentsku aplikaciju šalje poruku kontretnom korisniku ili grupi, preko WebSocket-a.

Kada poruka stigne na serversku stranu parsira se i ako je grupna treba je poslati svakom korisniku iz grupe. Ukoliko se neki korisnik nalazi na istom hostu kao i pošiljalac dovoljno je putem WebSocket-a proslediti poruku tom klijentu, u suprotnom potrebno je poruku proslediti odgovarajućem hostu koji će je dalje putem WebSocket-a proslediti krajnjem korisniku. Ukoliko neki korisnik nije aktivan nije mu potrebno direktno slati poruku, već se ona samo treba sačuvati u bazu. Kada se korisnik uloguje na sistem treba da vidi svoje propuštene poruke (ukoliko ih ima). **Trenutak u kom ćete sačuvati poruku u bazu je prepušten vama.**

CHAT APP – CLIENT SIDE

Klijentska aplikacija treba da bude rich client front end aplikacija, napisana u Angularu radnom okviru ili upotrebom React biblioteke. Sva komunikacija između klijenta i servera se vrši putem WebSocket-a, i

jedino se putem WebSocket poruka može promeniti stanje aplikacije.

Aplikacija treba da sadrži četiri segmenta, i to su forma za registraciju, forma za prijavu na sistem, profil korisnika i glavna čet stranica. Na profilnoj stranici korisnik treba da vidi svoje prijatelje, grupe u kojima se nalazi, kao i sve konverzacije koje je do sada imao (može i na čet stranici).

Dodavanju prijatelja prethodi pretraga postojećih korisnika. Obezbediti pretragu po imenu, prezimenu i korisničkom imenu.

Logout mehanizam implementirati tako da se može izvršiti eksplicitno (klikom na Logout dugme), ali i implicitno kada se zatvori WebSocket sesija (npr. kada se zatvori tab ili osveži stranica).