SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5191

Primjena paradigme poslužiteljskih dojava u razvoju web-usluga

Karlo Vrbić

Zagreb, lipanj 2017.

SADRŽAJ

POPIS SLIKA

POPIS KODOVA

1. Uvod

U 21. stoljeću svjedočimo sve bržem razvoju tehnologija u svim područjima, a najviše na području informacijskih tehnologija i mobilnih tehnologija. Danas je nezamisliv život bez mobilnih uređaja poput prijenosnih računala, pametnih mobitela, tableta, pametnih satova, itd. To potvrđuje podatak da 60% ljudi posjeduje mobilni telefon i da danas većina ljudi internetu pristupa pomoću mobilnih uređaja. Do tako drastičnog porasta broja korisnika mobilnih uređaja došlo je ponajviše razvojem tzv. pametnih mobitela. Danas tržištem dominiraju Android i iOS pametni mobilni uređaji koji korisniku pružaju mnoge mogućnosti koje nisu bile dostupne tradicionalnim mobilnim telefonima. Uz razvoj mobilnih mreža i bežičnog interneta korisnicima je omogućeno da budu spojeni na internet bez obzira na vrijeme i mjesto gdje se nalaze. Ti podatci drastično mijenjaju način na koji korisnici koriste svoje uređaje. Od uređaja se više ne zahtjeva samo da može pristupiti informacijama bitnih korisniku nego i da može primati informacije iz više izvora u tren kada te informacije budu dostupne. Standardna klijentsko-poslužnička arhitektura ne zadovoljava tim uvjeta i u tome je motivacija za razvoj tehnologija poslužiteljskih dojava.

2. Tehnologije poslužiteljskih dojava

2.1. Comet

U početcima World Wide Web tehnologije internet preglednici za svaku stranicu poslao više zahtjeva prema poslužitelju, svaki zahtjev za jednu komponentu stranice, npr. HTML dokument, CSS dokument, skripte ili bilo koji drugi resurs koji se upotrebljavao na stranici. Preuzimanje stranice u dijelove podataka poznato je kao model stranica po stranica model(engl. *Page-by-Page model*). Za dohvat novog sadržaja stranice stranica se trebala ponovo učitati.

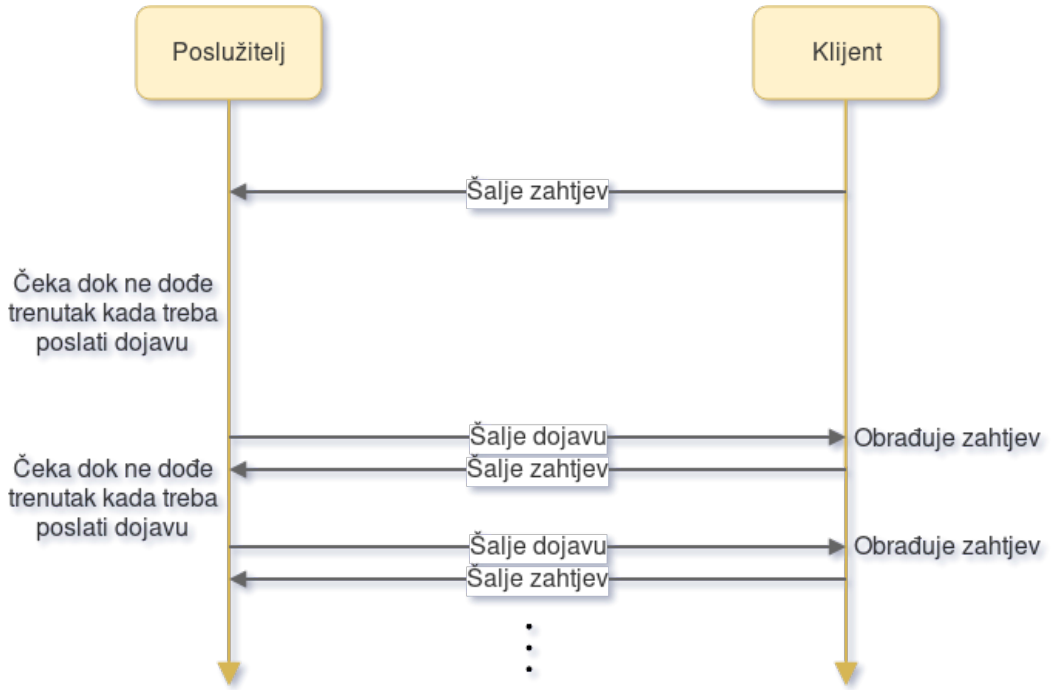
AJAX(Asynchronous Javascript and XML) tehnologija je donijela promjene i omogućila učitavanje dijelova stranice kroz asinkrone zahtjeve prema poslužitelju ali AJAX nije riješio sve probleme dinamičkih web stranica. Iako web preglednici sa AJAX-om imaju mogućnost zatražiti podatke sa servera i dinamički izmijeniti stranicu kada ti podatci stignu i dalje nisu imali mehanizam kojim bi znali da li server uopće ima nove podatke u slučajevima kada je to potrebno, npr. chat aplikacije. U takvim slučajevima se koristila tzv. „polling“ tehnika.(Gravelle, 2009)

2.1.1. Polling

„Polling“ tehnika je tehnika ostvarivanja poslužiteljskih dojava koja koristi periodičko slanje zahtjeva prema poslužitelju i na taj način doznaje da li poslužitelj ima relevantnih podataka koje bi klijent trebao prikazati. Ova tehnika ustvari nije prava „push“ tehnologija jer bi takva tehnologija omogućila poslužitelju slanje podataka klijentu bez zahtjeva klijenta. Ovu tehniku možemo podijeliti na dvije kategorije: „polling“ i „long polling“.

„Polling“ tehnika se obično ostvaruje tako da klijent ostvari vezu prema poslužitelju, pošalje HTTP zahtjeve prema poslužitelju, poslužitelj odgovara klijentu i šalje mu podatke sa dojavom ukoliko je dostupna te se nakon što klijent primi odgovor veza raskida. Ovaj proces bi se periodički ponavljao te bi tako korisnik naizgled imao prave

poslužiteljske dojave. Ova tehnika se obično koristila na internet stranicama pomoću periodičkih AJAX zahtjeva. Problemi ovakve implementacije poslužiteljskih dojava su očiti. Zbog potrebe za slanjem periodičkih HTTP zahtjeva koji su često i bespotrebnih poslužitelja se tjera na nepotrebnu obradu velikog broja zahtjeva što zahtjeva bolji i skuplji hardver ali i povećava internet promet zbog učestalih zahtjeva.



Slika 2.1: Dijagram razmjene dojava primjenom „long polling“ tehnike

Sustavi koji upotrebljavaju „polling“ tehniku su ne skalabilni jer veći broj korisnika drastično povećava broj zahtjeva koje server mora obrađivati što je skupo i ne održivo. Zbog navedenih problema sa normalnom „polling“ tehnikom razvijena je tehnika „long polling“. Ova tehnika je slična svom prethodniku ali se razlikuje u načinu na koji poslužitelj odgovara. Koristeći ovu tehniku klijent mora ostvariti vezu sa poslužiteljom, poslati HTTP zahtjev prema poslužitelju ali poslužitelj neće odmah odgovoriti(slika 2.1). Kod HTTP-a TCP veza se održava sve dok klijent ne dobije odgovor. Ova činjenica je ovdje iskorištena na način da će poslužitelj pričekati dok relevantni podatci ili dojave ne budu dostupne pa će tek onda kada je to potrebno poslati odgovor klijentu. Klijent nakon što zaprimi odgovor šalje novi zahtjev poslužitelju i proces se ponavlja. Ako se ipak zbog nekih razloga veza raskine prije nego što poslužitelj pošalje odgovor klijent pokušava ostvariti novu vezu i poslati novi zahtjev. Iako se ovom tehnikom rješava problem velikog broja zahtjeva stvara se novi problem. Većina modernih poslužitelja radi na način da za svaki zahtjev stvara novi proces ili dretvu i nakon obrade

zahtjeva prekida proces ili dretvu ili zahtjeve predaje dretvi iz bazena dretvi nakon obrade zahtjeva vraća dretvu u bazen dretvi. Problem je što koristeći ovu tehniku većinu vremena poslužitelj čeka podatke koje treba poslati a za to vrijeme ima proces ili dretvu koja ne radi ništa osim što koristi resurse poslužitelja koji bi se mogu koristiti za ostale zadatke. Još gori je problem ukoliko se koristi bazen dretvi jer može doći do situacije u kojoj sve dretve iz bazena čekaju podatke pa nema ni jedne dretve dostupne za obradu novih zahtjeva koji čekaju u redu ili se odbacuju.

2.1.2. Streaming

Pored „polling“ tehnika imamo i „streaming“ tehniku ostvarivanja poslužiteljskih dojava. Ova tehnika se ostvaruje pomoću jedne veze koju klijent ostvari sa poslužiteljem koju potom poslužitelj koristi za slanje dojava klijentu. Ako se odlučimo za implementaciju ove tehnike imamo dvije opcije: korištenje metode skrivenih iframe-ova ili korištenje višestrukog odgovora pomoću XMLHttpRequest objekta koji se koristi pri slanju AJAX zahtjeva.

„iframe“ je HTML element koji se koristi za ugradnju dokumenta unutar drugog HTML dokumenta. Pri implementaciji streaming Comet tehnike može se koristiti skriveni iframe element čiji će „src“ atribut pokazivati na URL s kojeg će poslužitelj slati dojave. Svaki put kada poslužitelj ima dojavu za klijenta on će tu dojavu zapakirati u Javascript skriptu koju će klijent dobiti kroz iframe element te će je potom izvršiti.(Carbou, 2011)

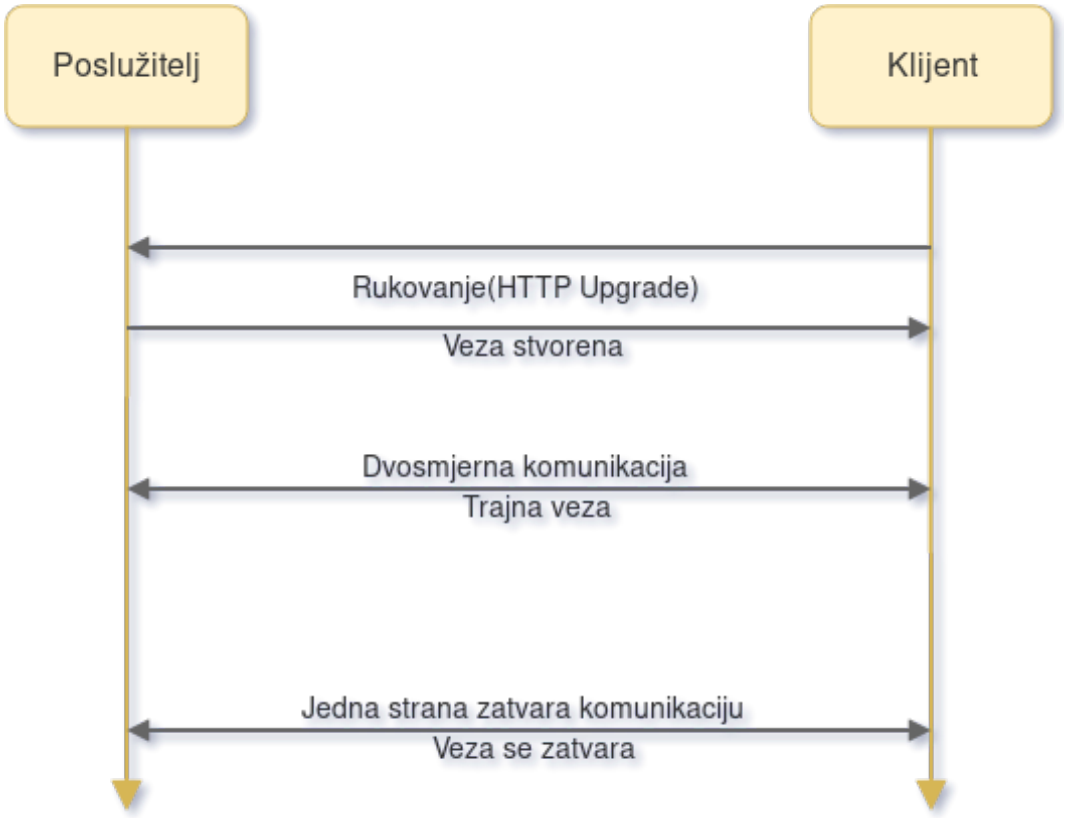
2.2. WebSocket

Povijesno, stvaranje web aplikacija koje trebaju dvosmjernu komunikaciju između klijenta i poslužitelja zahtijevala je korištenje HTTP-a na način na koji HTTP nije namijenjen. Jednostavnije riješenje bi bilo da se koristi jedna TCP veza. Upravo na tome je radila IETF(engl. Internet Engineering Task Force) i objavila pod RFC 6455 specifikacijom. Protokol se zove WebSocket i zajedno sa WebSocket API-ijem daje alternativu „pollingu“ i sličnim tehnikama. WebSocket protokol pokušava riješiti ciljeve postojećih dvosmjernih HTTP tehnologija u kontekstu postojeće HTTP infrastrukture. Kao takav, osmišljen je za rad preko HTTP portova 80 i 443, kao i za podršku HTTP posrednika.

Protokol trenutno podržavaju svi poznatiji internet preglednici poput Firefox(PC i Android verzija), Chrome(PC i mobilna verzija), Safari(PC i iOS verzija), Opera(PC i

mobilna verzija), Internet Explorer i Android browser.

Protokol se sastoji od dva dijela: rukovanja(engl. *handshake*) i prijenosa podataka.(Fette i Melnikov, 2011)



Slika 2.2: Dijagram razmjene dojava primjenom „WebSocket“ tehnologije

2.2.1. Rukovanje

Klijent tijekom rukovanja poslužitelju šalje zahtjev(kod 2.1) na koji poslužitelj odgovara(kod 2.2).

Klijent uključuje hostname u Host zaglavlje rukovanja prema tako da i klijent i poslužitelj mogu potvrditi da se slažu o tome koji je host u upotrebi.

Dodatna polja zaglavlja koriste se za odabir opcija u WebSocket protokolu. Tipične opcije dostupne u ovoj verziji su selektor podprotokola („Sec-WebSocket-Protocol“), popis ekstenzije podrške od strane klijenta („Sec-WebSocket-Extensions“), „Origin“ zaglavlje itd.

„Sec-WebSocket-Protocol“ zaglavlje zahtjeva može se upotrijebiti za označavanje podprotokola koje klijent podržava. Poslužitelj odabire jedan ili nijedan od prihvatlji-

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

Kod 2.1: Zahtjev klijenta tijekom rukovanja

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

Kod 2.2: Odgovor poslužitelja tijekom rukovanja

vih protokola i šalje tu vrijednost tijekom rukovanja kako bi naznačio da je odabrao taj protokol.

Rukovanje je slično HTTP-u dopuštajući poslužiteljima da obrađuju HTTP i WebSocket veze na istom portu. Kada se uspostavlja veza, komunikacija prelazi na dvosmjerni protokol koji ne odgovara HTTP protokolu. Uz „Upgrade“ zaglavlje klijent šalje i „Sec-WebSocket-Key“ zaglavlje koje sadrži nasumične oktet kodirane sa base64 algoritmom a poslužitelj odogovara sa hash vrijednosti tog ključa u „Sec-WebSocket-Accept“ zaglavlju.

2.2.2. Prijenos podataka

Nakon rukovanja i klijent i poslužitelj mogu poslati podatke u oba smjera. WebSocket prijenosi zovu se poruke, gdje se jedna poruka može po želji podijeliti na više okvira podataka. To može omogućiti slanje poruka gdje su početni podaci dostupni, ali cjelokupna duljina poruke je nepoznata (šalje se jedan podatkovni okvir nakon drugog dok se ne postigne kraj i označeno s bitom FIN). Uz proširenja protokola, to se također može koristiti za multipleksiranje nekoliko veza istovremeno.

Svaki okvir ima svoju vrstu ali svi okviri koji čine jednu poruku moraju biti iste vrste. Općenito govoreći, postoje vrste za tekstualne podatke (koji se tumače kao tekst kodiran UTF-8 kodnom stranicom), binarne podatke (čija interpretacija ostaje do apli-

kacije) i kontrolne okvire (koji nisu namijenjeni za prijenos podataka za aplikaciju Ali umjesto toga za signalizaciju na razini protokola, kao da signalizira da veza bude zatvorena). Ova verzija protokola definira šest vrsta okvira i ostavlja deset rezerviranih za buduću upotrebu.

2.3. Firebase Cloud Messaging (FCM)

2.3.1. Povijest

Pojavom Android pametnih mobilnih telefona Google, kompanija iza Androida, tražila je riješenje za omogućivanje dvosmjerne komunikacije između Android aplikacija i poslužitelja. Postojeća rješenja nisu bila dovoljno dobra za mobilne uređaje. Jedan od razloga je da korisnik mobilnog uređaja može od jednom imati veliki broj aplikacija koje zahtjevaju opciju poslužiteljskih dojava a postojeće tehnologije su koristile jednu vezu po aplikaciji. Problem je što su mobilni uređaji relativno slabiji od osobnih uređaja i nisu bili u mogućnosti koristiti veliki broj veza odjednom.

Prva verzija protokola nazvala se AC2DM(Android Cloud to Device Messaging). Protokol je omogućavao sve aplikacije primaju poslužiteljske dojave preko jedne veze sa Google-ovim poslužiteljem. GCM(engl. Google Cloud Messaging) usluga najavljena je u lipnju 2012. kao nasljednik protokola Android Cloud to Device Messaging. Poboljšanja su uključivala poboljšanja za provjeru autentičnosti i isporuke, nove krajnje točke API-ja i parametre za razmjenu poruka, te uklanjanje ograničenja na broj odaslanih poruka i veličine poruka. 2014 godine Google preuzima tvrtku Firebase i pod imenom Firebase Cloud Messaging izbacuje protokol koji je sada dostupan na Android i iOS pametnim mobilnim uređajima ali i na klasičnim web aplikacijama.

Slika 2.3: Princip rada Firebase Cloud Messaging servisa

2.3.2. Prijava na uslugu

Firebase Cloud Messaging protokol radi na način da klijent koji na svom uređaju ima barem jednu aplikaciju koju koristi FCM protokol mora ostvariti vezu sa FCM poslužiteljem(slika 2.3 postupak 1). Na Android uređajima svaka aplikacija mora imati servis koji će inicirati vezu sa FCM poslužiteljem i dobiti token jedinstven za svaku

```
{
    "to" : "APA91bHun4MxP5egoKMwt2...",
    "notification" : {
      "body" : "Ovdje se nalazi tijelo dojave",
      "title" : "Ovdje se nalazi naslov dojave",
      "icon" : "Ovdje se nalazi ime ikone koja će se prikazati uz dojavu"
    },
    "data" : {
      "example1" : "value1",
      "example2" : "value2"
    }
}
```

Kod 2.3: Primjer JSON objekta kojeg poslužitelj šalje FCM poslužitelju za slanje dojave

aplikaciju(slika 2.3 postupak 2). Nakon što klijent dobije token on ga šalje svom poslužitelju koji bi trebao token spremiti ukoliko planira slati dojave tom klijentu(slika 2.3 postupak 3).

Klijent isto tako ima mogućnost prijave na razne teme. Teme su skupine klijenata koji žele primati iste dojave a svaka tema se identificira pomoću svog jedinstvenog imena. FCM poslužitelj pamti koji klijenti su se pretplatili na koje teme te koriste tu informaciju prilikom slanja dojave.

2.3.3. Slanje dojava i podataka

Kada poslužitelj poželi poslati dojavu korisniku on za to koristi token koji je prije toga dobio i FCM poslužitelje. Na FCM poslužitelj šalje JSON objekt preko HTTP ili XMLL protokola. JSON objekt se sastoji od dva atributa: „to“ ili „condition“, „notification“ i „data“ (slika 2.3 postupak 4), a zatim FCM poslužitelj obrađuje dojavu, sprema u svoju bazu podataka ako klijent nije spojen ili šalje dojavu klijentu(slika 2.3 postupak 5) sa JSON objektom(kod 2.3) kao tijelom poruke.

Prvi atribut može biti „to“ ili „condition“ i oba služe za određivanje klijenta ili više njih kome će slati dojava. Kod atributa „to“ to može biti token koji je prije toga klijent poslao svom poslužitelju ili može biti ime teme. Vrijednost atributa „condition“ sadrži neke uvijete koje primatelji moraju ispunjavati, npr. „"condition": "’dogs’ in topics || ’cats’ in topics"“ znači da će se dojava poslati klijentima koji su pretplaćeni na teme „dogs“ ili „cats“ .U oba slučaja FCM poslužitelj koristi ovaj podatak da razluči kome treba poslati dojavu.

Atribut „notification“ se koristi za slanje dojava koje će se prikazati u dijelu za dojave na klijentskom operacijskom sustavu. Ovaj atribut je objekt koji se sastoji od atributa poput „body“, „title“, „icon“, itd. koji određuju informacije koje će se prenijeti ali i koja će se ikonica prikazati uz dojavu ili koji zvuk će se oglasiti prilikom primitka dojave.

Atribut „data“ se koristi ukoliko se treba poslati veća količina podataka. Pomoću ovog atributa može se poslati do 4kB podataka u obliku ključ-vrijednost. iOS klijent će ove podatke primiti tek kada klijentska aplikacija dođe u prednji plan dok će Android aplikacija obraditi podatke i kad je u pozadini. U slučaju ako se šalju i „notification“ i „data“ atribut dojava će se prikazati i ako je aplikacija u pozadini dok će se „data“ atribut obraditi tek kada aplikacija dođe u prednji plan.(fcm, 2017)

2.4. Apple Push Notification Service

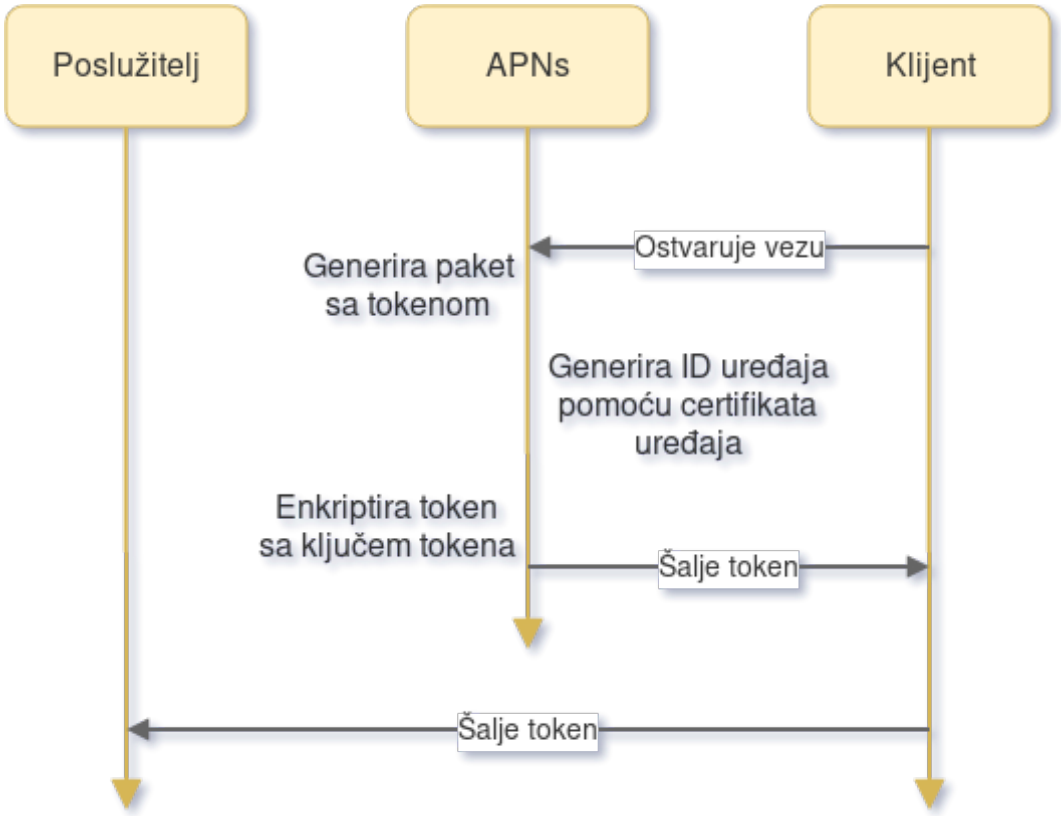
Prvi pametni mobilni uređaj iPhone kompanije Apple pokrenuo je revoluciju u razvoju mobilnih uređaja i izmjenio način na koji korisnici koriste svoje mobilne uređaje. Veliki broj aplikacija se oslanjao na poslužiteljske dojave ali nije bilo adekvatnog rješenja. Apple se susreo sa istim problemima kao i njihov konkurent Google pa su se odlučili za izradu vlastitog servisa za poslužiteljske dojave.

2008 godine napravljen je servis nazvan APNs(engl. Apple Push Notification Service). Usluga podržava iOS pametne mobilne telefone, watchOS pametne satove, tvOS pametne televizore i macOS osobna i prijenosna računala.

Princip rada je sličan Firebase Cloud Messaging usluzi. Klijenti se registriraju kod APNs poslužitelja i APNs vodi računa o vezama sa klijentima. Poslužitelj kada poželi poslati dojavu šalje zahtjev prema APNs poslužitelju koji šalje te dojave klijentima.

2.4.1. Prijava na uslugu

Po početnom pokretanju aplikacije na korisničkom uređaju sustav automatski uspostavlja akreditiranu, šifriranu i postojanu IP vezu između aplikacije i APNs poslužitelja. Ta veza omogućuje vašoj aplikaciji da izvrši postavljanje kako bi omogućila primanje dojava. APNs poslužitelj nakon stvaranja veze sa klijentom generira paket sa token, generira ID uređaja pomoću certifikata uređaja te još enkriptira token sa ključem tokena. Ako je proces autentikacije uspješno izvršen klijentska aplikacija dobije token. Svaki token je jedinstven za aplikaciju i uređaj koji korisnik koristi(slika 2.4).



Slika 2.4: Upravljanje tokenom uređaja koristeći APNs

2.4.2. Slanje dojava i podataka

Poslužitelj pri slanju dojave šalje zahtjev APNs poslužitelju preko HTTP/2 protokola. U tijelu zahtjeva treba poslati JSON objekt koji ne prelazi veličinu od 4kB ili VoIP poruku koja ne prelazi 5kB.

JSON objekt se sastoji od atributa „aps“ i drugih atributa koji nemaju određeno ime a koriste su za prijenos podataka u obliku ključ-vrijednost. Vrijednost atributa „aps“ je objekt koji sadrži atribute „title“, „body“ , „alert“, „sound“, itd. koji određuju sadržaj notifikacije i način prikaza. Ako se želi poslati tzv. „tiha“ dojava pošalje se posebno oblikovani zahtjev(kod 2.4).

Tiha dojava označava aplikaciji da ima dostupne dojave i da ih može dohvatiti kada to poželi.(apn, 2017)

```
{
    "aps" : {
        "content-available" : 1
    }
}
```

Kod 2.4: JSON objekt za „tihu“ dojavu

3. Arhitektura aplikacije

3.1. Ideja

Ideja je napraviti aplikaciju koja korisniku omogućava praćenje događaja poput koncerata, rođendana i sl. i informacija vezanih uz taj događaj. Isto tako korisnici trebaju imati mogućnost kreiranja događaja za koje mogu definirati nekakve atribute poput naziva događaja, opisa, datuma i sl. Svi pratioci koji prate određeni događaj bi trebali dobiti dojavu kada se kreator događaja odluči izmijeniti neki od atributa događaja ili odluči poslati poruku vezanu uz taj događaj. Tako će se dobiti aplikacija koja će biti koristan alat pri organizaciji manjih događaja ali i većih događaja. Aplikacija se lako može pretvoriti u manju društvenu mrežu na kojoj će fokus biti na događajima, a ne na objavama kao većina društvenih mreža. Aplikacija se sastoji od poslužiteljskog dijela i klijentskog dijela.

3.2. Protokol za razmjenu podataka

Za funkcioniranje aplikacije mora se implementirati protokol za razmjenu podataka između poslužitelja i klijenta. Većina razmjene podataka u ovoj aplikaciji otpada na klijentske zahtjeve i poslužiteljske odgovore i stoga će se koristiti HTTP/1.1 koji će u svom tijelu sadržavati poruku u JSON formatu. HTTP i JSON su standardne tehnologije kada su u pitanju slične aplikacije poput ove jer je prijenos podataka jednostavan i ne koristi puno internetskog prometa.

3.3. Autentikacija

Aplikacija treba imati mogućnosti registracije i prijave kako bi raspoznala korisnike i pamtila njihove odabire. Svaki korisnik bi trebao imati jedinstveno korisničko ime. Prilikom otvaranja klijentske aplikacije korisnika se mora zatražiti da unese svoje

korisničko ime i lozinku. Ako se radi o novom korisniku mora mu se ponuditi mogućnost registracije, a pri registraciji korisnici bi trebali unijeti svoje osobne podatke, korisničko ime njihovog budućeg računa i lozinku. Nakon što unese korisničko ime i lozinku korisnik pomoću klijentske aplikacije šalje zahtjev prema poslužitelju koji će utvrditi da li se radi o ispravnim podacima. Ako se radi o ispravnim podacima poslužitelj će izračunati „token“ koristeći OAuth2 standard. Tijekom svoje komunikacije sa poslužiteljem, klijent će koristiti dobiveni token kako bi poslužitelj svaki put mogao provjeriti da li se zaista radi o pravom korisniku.

3.4. Spremanje podataka

Poslužiteljski dio je odgovoran za spremanje svih podataka vezanih uz aplikaciju. Za spremanje podataka koristiti ćemo relacijske baze podataka, a za lakši rad i lakše pristupanje bazi podataka koristiti ćemo objektno-relacijsko mapiranje(engl. object-relational mapping).

3.5. Poslužiteljske dojave

Poslužiteljske dojave se ovdje koriste kao način na koji se obavješćuje pretplatnika o izmjenama u događajima koje prate. Pošto izmjene mogu biti veoma bitne korisniku vrlo je bitno da se poruke dostave u što kraćem roku ali i da se ne izgube ukoliko klijent nije spojen na poslužitelj u tom trenutku(npr. korisnikov uređaj nema pristup internetu ili je ugašen).

4. Implementacija

4.1. Poslužiteljske dojave

Prilikom odabira tehnologije za poslužiteljske dojave velika pažnja je bila na brzini slanja dojava i načinu na koji određena tehnologija rješava problem korisnika koji nije spojen na poslužitelj u trenutku dolaska dojave. Isto tako bitno je koje sve platforme ta tehnologija podržava jer bi u budućnosti mogli poželjeti izraditi klijentsku aplikaciju za neku drugu platformu.

Odabrao sam Firebase Cloud Messaging servis. Firebase Cloud Messaging pruža mogućnost brzog slanja dojava, a ako dojavu nije moguće poslati u trenutku kada se dojava pojavila(npr. korisnik nije spojen na internet ili je ugasio uređaj) FCM servis će automatski spremiti dojavu i poslati je čim korisnik ponovo ostvari vezu sa FCM poslužiteljem. Velika prednost Firebase-ovog servisa je ta da je više-platformski. FCM podržava Android i iOS operativne sustave ali i klasične web aplikacije.

4.2. Klijent

Pri implementaciji klijenta odlučio sam se za Android platformu. Prednost naspram klasičnih web aplikacija je ta da klijent može imati stalne pozadinske procese koji će biti u mogućnosti primati dojave i kad korisnik ne koristi aplikaciju dok je prednost naspram iOS platforme ta da se pametni mobilni telefoni sa iOS operativnim sustavom manje koriste nego Android telefoni. U 4. kvartetu 2016. godine od svih prodanih mobilnih uređaja 81.7% (Vincent, 2017) su bili pametni mobilni telefoni sa Android operativnim sustavom. Time Android dobiva prioritet pri izradi klijentske aplikacije.

Za spajanje na poslužitelja koristio sam dvije knjižnice: OkHttp i Retrofit2. OkHttp je HTTP klijent koji uz mogućnost slanja HTTP zahtjeva koristi i bazen veza za smanjenje latencije zahtjeva, cache odgovora za sprečavanje ponovljenih zahtjeva

te podržava HTTP/2 što je korisno ukoliko se bude mijenjao protokol u budućnosti. Retrofit2 je knjižnica koja se koristi za jednostavniju uporabu oblikovnog obrazca repozitorij kod repozitorija koji koristi HTTP vezu.

Za svaku aplikaciju vrlo bitno je da je responzivna u svim trenucima. Kako bi rješavanju tog problema pristupili što lakše koristio sam knjižnicu RxJava2 koja nam omogućuje tzv. „Reactive“ programiranje sa Java programskim jezikom. RxJava2 nam omogućuje da lako upravljamo dretvama i paraleliziramo poslove koje obavlja aplikacija. Da bi aplikacija bila brza i responzivna sve poslove koji bi mogli malo dulje trajati je poželjno raditi na posebnoj dretvi.

Da bi omogućili klijentu vezu sa Firebase Cloud Messaging poslužiteljem moramo koristiti Android knjižnicu za FCM pomoću koje sam napravio servis koji će biti obaviješten kada korisnik primi dojavu. Taj servis će pri primljenoj dojavi, tu dojavu obraditi i na kraju obavijestiti korisnika sa obavijesti u traci za dojave.

4.3. Poslužitelj

Pri implementaciji poslužiteljskog dijela aplikacije koristio sam Java EE tehnologije zbog toga što su Java programi više-platformski, ali i zbog česte uporabe ove platforme u implementaciji velikih sustava za razne kompanije. Zbog dobre zastupljenosti danas za u ovoj tehnologiji za neke česte probleme postoje dobra i isprobana rješenja što uvelike smanjuje količinu rada sa moje strane.

Za prihvaćanje zahtjeva sa klijenta i razvoj API-a koristio sam Jersey. Jersey je implementacija JAX-RS(Java API for RESTful Web Services) specifikacije koja je napravljen za lakšu implementaciju REST i sličnih API-a. Ovom metodom ne moramo ništa eksplicitno podešavati osim što moramo označiti koja metoda će se pokrenuti za koji tip zahtjeva.

Spremanje podataka i pretraga tih podataka je najvažnija zadaća poslužiteljskog dijela programa. Za bazu podataka sam odabrao MySQL zbog toga što je to besplatna baza podataka otvorenog koda. MySQL je veoma popularna baza podataka i stoga ima veliku potporu zajednice pa je stoga primamljiv odabir naspram drugih besplatnih rješenja. Za rad sa bazom podataka sam koristio knjižnicu za objektno-relacijsko mapiranje Hibernate. Hibernate je implementacija JPA(Java persistency API) specifikacije koja koristi Java objekte za kreiranje tablica i spremanje podataka. Uz osnovne funkcionalnosti objektno-relacijsko mapiranja Hibernate podržava laku integraciju sa sustavima za cache-iranje poput EhCache-a. EhCache će spremati pozive prema bazi podataka u memoriju tako da će uvelike ubrzati rad sa bazama podataka koji često zna

usporiti ostatak aplikacije.

Prilikom interakcije sa klijentom bitno je da poslužitelj može pouzdano utvrditi identitet korisnika koji šalje upite i stoga sam koristio Apache Oltu knjižnicu. Apache Oltu podržava OAuth i OAuth2 protokole i ova knjićnica je veoma jednostavna za korištenje.

5. Zaključak

U ovom završnom radu predstavljena su tehnologije za poslužiteljske dojave, od starijih rješenja koja su imitirala poslužiteljske dojave poput „long polling-a“ do novih tehnologija poput „WebSocket-a“ i servisa poput „Firebase Cloud Messaging-a“. Prikazano je kako se prvo problem poslužiteljskih dojava pokušao riješiti raznim domišljatim metodama korištenja HTTP-a, dok moderna rješenja uključuju nove protokole koji koriste TCP vezu za slanje u oba smjera.

Prije izrade ovog rada malo toga sam znao o tome kako funkcioniraju poslužiteljske dojave i koje se tehnologije koriste. Tijekom pripreme i izrade ovog završnog rada naučio sam dosta o prijašnjim metodama imitacije poslužiteljskih dojava, ali i o modernim tehnologijama i uslugama. Osim učenja kroz proučavanje tehnologija sa „Firebase Cloud Messaging-om“ sam se imao prilike više upoznati kroz praktični rad gdje sam paradigmu poslužiteljskih dojava koristio za dojavljivanja promjena na korisniku važnim događajima.

LITERATURA

Local and remote notification programming guide: Apns overview, ožujak 2017. URL https://developer.apple.com/go/?id=push-notifications. 31. svibnja 2017.

Firebase cloud messaging | firebase, svibanj 2017. URL https://firebase.google.com/docs/cloud-messaging/. 31. svibnja 2017.

Mathieu Carbou. Reverse ajax, part 1: Introduction to comet, srpanj 2011. URL https://www.ibm.com/developerworks/library/wa-reverseajax1/. 31. svibnja 2017.

Ian Fette i Alexey Melnikov. The websocket protocol. RFC 6455, RFC Editor, December 2011. URL https://tools.ietf.org/html/rfc6455.

Rob Gravelle. Comet programming: Using ajax to simulate server push. *Webreference tutorial, Mar*, 2009.

James Vincent. 99.6 percent of new smartphones run android or ios — the verge, veljača 2017. URL https://www.theverge.com/2017/2/16/14634656/android-ios-market-share-blackberry-2016. 31. svibnja 2017.

Primjena paradigme poslužiteljskih dojava u razvoju web-usluga

Sažetak

Dolazak pametnih mobilnih telefona ali i drugih prijenosnih uređaja uvelike je promijenio način korištenja aplikacija. Korisnici danas zahtjevaju da informacija dolazi do njih čim bude dostupna umjesto da je oni moraju zahtijevati. Kroz ovaj rad predstavljeno je par načina na koji je se ovaj problem pokušao riješiti. Jedan od načina, Firebase Cloud Messaging, sam implementirao na Android platformi kako bi bolje shvatio način rada i integracije takvih i sličnih sustava u vlastitu aplikaciju.

Ključne riječi: Ključne riječi, odvojene zarezima.

Application of Server Push Notifications in Web Service Development

Abstract

The arrival of smartphones and other portable devices has greatly changed the way the apps are used. Users today require information to be available as soon as it is available instead of having to request it manually. This work presents a few ways in which this problem has been tackled. I've implemented one of those services, Firebase Cloud Messaging, on an Android platform to better understand the way you work and integrate such and similar systems into your own application.

Keywords: Keywords.