

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	vi
1. Uvod	1
2. Tehnologije poslužiteljskih dojava	2
2.1. Comet	2
2.1.1. Prozivanje	3
2.1.2. Blokirajuće prozivanje	3
2.1.3. Strujanje	5
2.2. WebSocket	8
2.2.1. Rukovanje	8
2.2.2. Prijenos podataka	9
2.3. Firebase Cloud Messaging (FCM)	10
2.3.1. Prijava na uslugu	10
2.3.2. Slanje dojava i podataka	12
2.4. Apple Push Notification Service	13
2.4.1. Prijava na uslugu	14
2.4.2. Slanje dojava i podataka	14
3. Arhitektura sustava za praćenje događaja	15
3.1. Protokol za razmjenu podataka	15
3.2. Autentikacija	16
3.3. Spremanje podataka	17
3.4. Poslužiteljske dojave	17
4. Programsko ostvarenje sustava za praćenje događaja	18
4.1. Poslužiteljske dojave	18
4.2. Klijent	18
4.2.1. Prijava korisnika	19

4.2.2. Registracija korisnika	19
4.2.3. Glavni prozor	21
4.3. Poslužitelj	23
5. Zaključak	27
Literatura	28

POPIS SLIKA

2.1. Dijagram razmjene dojava primjenom tehnike prozivanja	3
2.2. Dijagram razmjene dojava primjenom tehnike blokirajućeg prozivanja	4
2.3. Dijagram razmjene dojava primjenom strujajuće <i>Comet</i> tehnike	5
2.4. Primjer korištenja <code>EventSource</code> objekta za prikaz poslužiteljskih dojava	7
2.5. Primjeri SSE poruka	7
2.6. Dijagram razmjene dojava primjenom tehnologije <i>WebSocket</i>	8
2.7. Zahtjev klijenta tijekom rukovanja	9
2.8. Odgovor poslužitelja tijekom rukovanja	9
2.9. Princip rada Firebase Cloud Messaging usluge	11
2.10. Primjer JSON objekta kojeg poslužitelj šalje FCM poslužitelju za sla- nje dojava	12
2.11. Upravljanje tokenom uređaja koristeći APNs	13
2.12. JSON objekt za „tihu“ dojavu	14
3.1. Slanje događaja i pretplata na događaj	16
3.2. Prikaz arhitekture sustava za praćenje događaja	17
4.1. Prozor za prijavu	20
4.2. Prozor za registraciju	20
4.3. Glavni prozor	21
4.4. Prozor za prikaz profila korisnika	22
4.5. Prozor za prikaz profila događaja	23
4.6. ER diagram poslužiteljske baze podataka	23
4.7. Primjer zahtjeva za registraciju	25

1. Uvod

U 21. stoljeću tehnologije u svim područjima se sve brže razvijaju, 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 pametni mobilni uređaji sa Android i iOS operativnim sustavima 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-poslužiteljska arhitektura ne zadovoljava tim uvjeta i u tome je motivacija za razvoj tehnologija poslužiteljskih dojava.

2. poglavlje govori o razvoju tehnologija poslužiteljskih dojava od *Comet* tehnologija pa do novijih tehnologija poput protokola *WebSocket* i usluge *Firebase Cloud Messaging* i *Apple Push Notification Service*. U tom poglavlju objasniti ćemo način rada pojedine tehnologije te prednosti i mane.

Slijedeće, 3. poglavlje govori o arhitekturi aplikacije za dojavu poruka i promjena na korisniku zanimljivih događaja. Za razvoj aplikacije definirati ćemo podjelu na klijentski i poslužiteljski dio aplikacije te ćemo definirati način razmjene podataka između njih. U ovom poglavlju dotaknuti ćemo se i bitnih dijelova aplikacije poput autentikacije i sustava za poslužiteljske dojave.

U 4. poglavlju preciznije se objašnjava rad programskog ostvarenja izrađenog u sklopu ovog rada te raspraviti o korištenim tehnologijama. Također se govori o mogućnostima ove aplikacije i načinu kako korisnik može koristiti ovaj sustav za praćenje raznih događaja ali i organiziranje vlastitih događaja.

2. Tehnologije poslužiteljskih dojava

Sredinom 90-ih godina prošlog stoljeća počeo je razvoj sustava koji podržavaju poslužiteljske dojave. Isprva, zbog nedostatka već gotovih i praktičnih rješenja, poslužiteljske dojave su se izvodile nekonvencionalnim korištenjem protokola HTTP, izvorno namijenjenog uporabi gdje klijent šalje zahtjev i dobija odgovor. Te tehnologije su se nazivale *Comet* tehnologije. U novije vrijeme razvile su se nove tehnologije poput protokola *WebSocket*, ali i usluga poput *Firebase Cloud Messaging* i *Apple Push Notification Service* koji sadrže više od same usluge poslužiteljskih dojava.

2.1. Comet

U počecima 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.

Asynchronous Javascript and XML (AJAX) 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. tehnika prozivanja.(Gravelle, 2009)

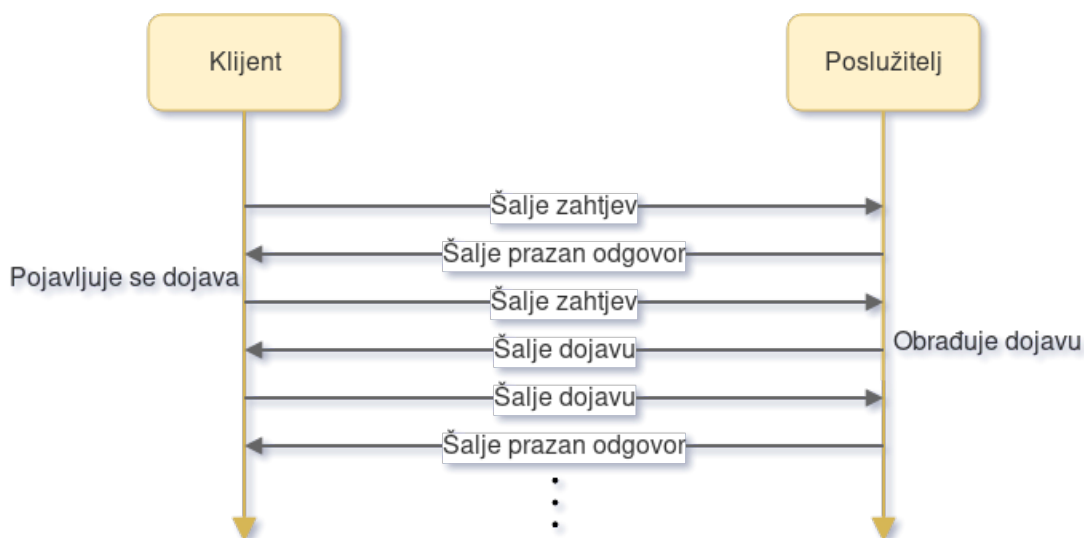
Comet je model web aplikacija u kojem se koristi HTTP veza za imitaciju poslužiteljskih dojava. U ovom modelu HTTP veza se održava živom kako bi se kroz tu vezu slale poslužiteljske dojave. Comet je u biti grupni naziv za sve tehnologije koje ostvaruju poslužiteljske dojave pomoću HTTP veze.

2.1.1. Prozivanje

Tehnika prozivanja (engl. *polling*) je tehnika ostvarivanja poslužiteljskih dojava koja koristi periodičko slanje zahtjeva prema poslužitelju i na taj način doznaje ima li poslužitelj relevantnih podataka koje bi klijent trebao dobiti. Ova tehnika ustvari nije prava tehnologija poslužiteljskih dojava jer bi takva tehnologija omogućila poslužitelju slanje podataka klijentu bez zahtjeva klijenta dok to ovdje nije slučaj.

Rad poslužiteljskih dojava ostvarene ovom tehnikom prikazan je na slici 2.1. Tehnika prozivanja 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 dojava. Ova tehnika se obično koristila na internetskim stranicama pomoću periodičkih AJAX zahtjeva.

Problemi ovakve implementacije poslužiteljskih dojava su očiti. Zbog potrebe za slanjem periodičkih HTTP zahtjeva, koji su često i bespotrebni, poslužitelja se tjera na nepotrebnu obradu velikog broja zahtjeva što zahtjeva bolji i skuplji hardver ali i povećava Internet promet zbog većeg broja zahtjeva.



Slika 2.1: Dijagram razmjene dojava primjenom tehnike prozivanja

2.1.2. Blokirajuće prozivanje

Zbog navedenih problema sa normalnom tehnikom prozivanja razvijena je tehnika blokirajućeg prozivanja. Ova tehnika je slična svom prethodniku ali se razlikuje u

Koristeći ovu tehniku klijent mora ostvariti vezu sa poslužiteljom, poslati HTTP zahtjev prema poslužitelju ali poslužitelj neće odmah odgovoriti. Kod protokola HTTP 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.



Sustavi koji upotrebljavaju tehniku blokirajućeg prozivanja su ne skalabilni jer veći broj korisnika drastično povećava broj zahtjeva koje server mora obrađivati što je skupo

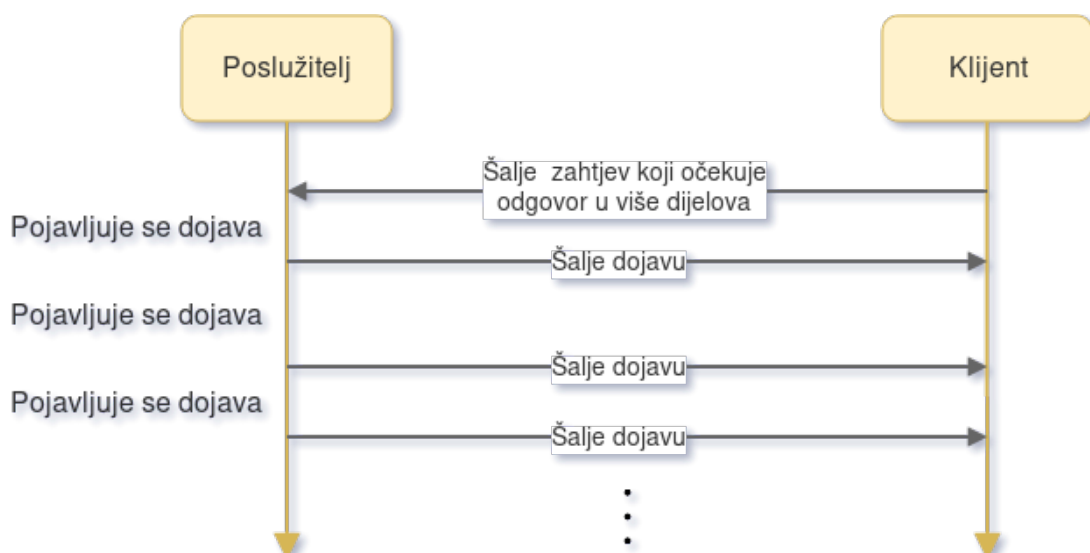
i ne održivo.

2.1.3. Strujanje

Pored tehnike prozivanja i blokirajućeg prozivanja imamo i tehniku strujanja. 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.

Drugi način se ostvaruje koristeći AJAX zahtjev koji očekuje višestruki odgovor, tj. odgovor u više dijelova. Uobičajeno bi poslužitelj poslao odgovore u više dijelova i u zadnjem označio da je riječ o zadnjem što bi dovelo do zatvaranja veze. Kod ove tehnike se koristi činjenica da se veza ne zatvara u koliko ne dođe do odgovora u kojem poslužitelj kaže da je zadnji odgovor i stoga može koristiti vezu za neograničeno slanje poslužiteljskih dojava klijentu. Na slici 2.3 je opisan rad ove tehnike. (Carbou, 2011)



Slika 2.3: Dijagram razmjene dojava primjenom strujajuće *Comet* tehnike

Server Sent Events

Poslužiteljske dojave pomoću AJAX zahtjev koji očekuje višestruki odgovor su dobro rješenje ali nedostaje precizna specifikacija koja bi opisala postupak implementacije takvog sustava. Skupina *Web Hypertext Application Technology Working Group* (WHATWG) ovaj problem je riješila sa svojim predloškom *Web Application 1.0*. Nakon što je web preglednik *Opera* implementirala ovu specifikaciju u listopadu 2006. godine ona se počinje nazivati *Server Sent Events* (SSE).

Upotreba ove specifikacije umjesto da se imitira pomoću Javascript objekta *XMLHttpRequest* ili HTML oznake *iframe* omogućuje klijentskoj aplikaciji bolju upotrebu mrežnih resursa, a to može rezultirati značajnim uštedama u trajanju baterije na prijenosnim uređajima.

Sučelje MessageEvent

Sučelje *MessageEvent* se koristi za poruke u velikom broju protokola poput *Server Sent Events*, *WebSocket*, *cross-document messaging*, *channel messaging* i *broadcast channels*. Sastoji se od pet ne promjenjivih atributa `data`, `origin`, `lastEventId`, `source` i `ports` prikazanih u tablici 2.1.

Tablica 2.1: Atributi sučelja *MessageEvent*

Atribut	Opis
<code>event.data</code>	Sadrži poruku koja je poslana
<code>event.origin</code>	Podrijetlo poruke (obično shema, host i port ali ne i putanja)
<code>event.lastEventId</code>	Id zadnjeg događaja
<code>event.source</code>	Ne koristi se za SSE
<code>event.ports</code>	Ne koristi se za SSE

Poruke

Korištenje ove specifikacije u klijentskoj web aplikaciji je vrlo jednostavno. Primjer korištenja je prikazan na slici 2.4. Kako bi mogli primiti poruke treba nam *Javascript* objekt `EventSource` kojeg stvaramo tako što njegovom konstruktoru predamo URL sa kojeg klijentska aplikacija treba primiti poslužiteljske dojave. Za svaku poruku koju klijent primi pozove se funkcija predana `onmessage` atributu koja je odgovorna za obradu poruke.

```
var source = new EventSource('updates.cgi');
source.onmessage = function (event) {
    alert(event.data);
};
```

Slika 2.4: Primjer korištenja `EventSource` objekta za prikaz poslužiteljskih dojava

```
event: add
data: 73857293

event: remove
data: 2153

event: add
data: 113411
```

Slika 2.5: Primjeri SSE poruka

Nakon poziva konstruktora objekta `EventSource` šalje se zahtjev poslužitelju sa predanim URL-om, a poslužitelj mu odgovara sa HTTP 200 OK odgovorom koji sadrži zaglavlje `Content-Type: text/event-stream`. Nakon toga klijent najavljuje vezu tako što postavlja atribut `readyState` na `OPEN` i šalje događaj naziva `open`. Nakon toga poslužitelj može početi sa slanjem poruka.

Primjere poruka možemo vidjeti na slici 2.5 gdje je prikazano 3 poruke. Svaki redak poruka je kodirana sa UTF-8 kodnom stranicom i može se sastojati od više dijelova kao što se u primjeru na slici 2.5 prikazane poruke od dva dijela. Svaki redak se interpretira na način da tekst prije znaka `:` označava polje (npr. `data`, `event`, `id` ili `retry`). Drugi dio događaja označava vrijednost tog polja. Ukoliko redak ne sadrži znak `:` onda se vrijednost tog polja tretira kao prazni tekst. Ukoliko imamo prazan redak onda treba odaslati dojavu sa svim prije napisanim redovima.

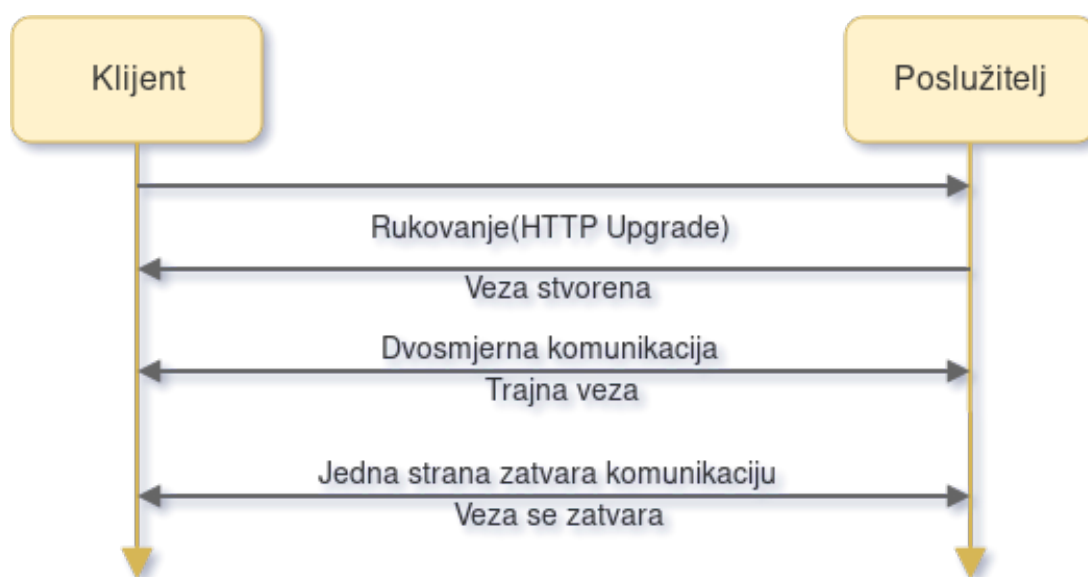
Svako ime polja ima posebnu namjenu. Ime polja `event` to označava tip poruke, `data` dodaje svoju vrijednost u `data` atribut, `id` postavlja `lastEventId` atribut a polje sa imenom `retry` postavlja vrijeme ponovnog povezivanja na vrijednost tog polja koje mora biti cjelobrojni broj. (WHATWG, 2017)

2.2. WebSocket

Povijesno, stvaranje web aplikacija koje trebaju dvosmjernu komunikaciju između klijenta i poslužitelja zahtijevala je korištenje protokola HTTP na način na koji HTTP nije namijenjen. Jednostavnije rješenje bi bilo da se koristi jedna TCP veza za slanje svih zahtjeva. Upravo na tome je radila *Internet Engineering Task Force* (IETF) skupina i objavila pod RFC 6455 specifikacijom. Protokol se zove WebSocket i zajedno sa WebSocket API-ijem daje alternativu tehnici prozivanja 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 2.6: Dijagram razmjene dojava primjenom tehnologije *WebSocket*

2.2.1. Rukovanje

Klijent tijekom rukovanja poslužitelju šalje zahtjev prikazan na slici 2.7 na koji poslužitelj odgovara odgovorom prikazanom na slici 2.8. Klijent uključuje naziv poslužitelja u `Host` zaglavlje rukovanja prema tako da i klijent i poslužitelj mogu potvrditi

```
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
```

Slika 2.7: Zahtjev klijenta tijekom rukovanja

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

Slika 2.8: Odgovor poslužitelja tijekom rukovanja

da se slažu o tome koji poslužitelj je u upotrebi.

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

Zaglavlje zahtjeva `Sec-WebSocket-Protocol` može se upotrijebiti za označavanje podprotokola koje klijent podržava. Poslužitelj odabire jedan ili nijedan od prihvatljivih protokola i šalje tu vrijednost tijekom rukovanja kako bi naznačio da je odabrao taj protokol.

Rukovanje je slično protokolu HTTP 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 zaglavlje `Upgrade` klijent šalje i zaglavlje `Sec-WebSocket-Key` koje sadrži nasumične oktet kodirane sa *Base64* algoritmom, a poslužitelj odgovara 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 aplikacije) 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.

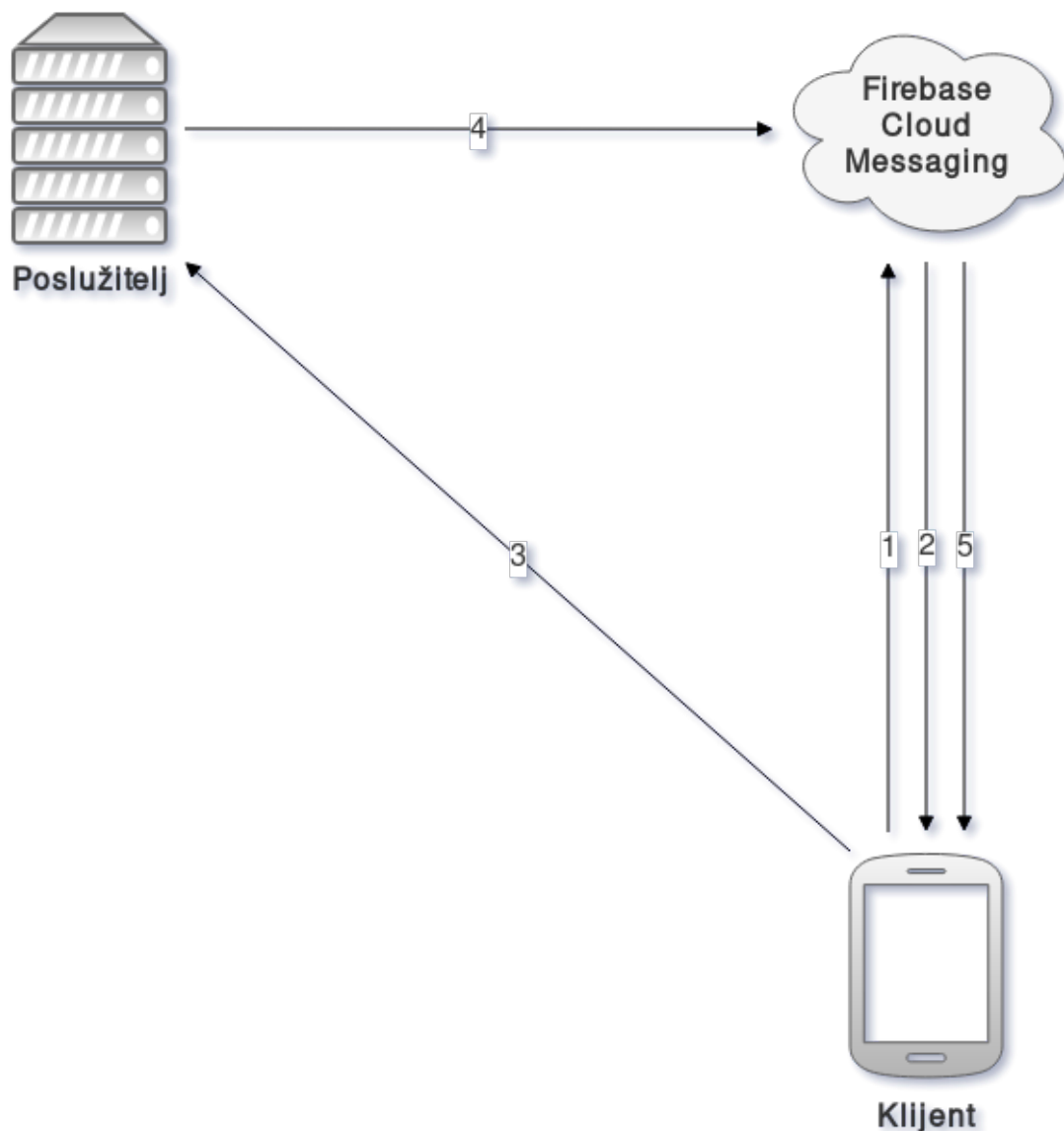
2.3. Firebase Cloud Messaging (FCM)

Pojavom pametnih mobilnih telefona sa Android operativnim sustavom Google, kompanija koja razvija Android operativni sustav, tražila je rješenje 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 zahtijevaju 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 *Android Cloud to Device Messaging* (AC2DM). Protokol je omogućavao sve aplikacije primaju poslužiteljske dojave preko jedne veze sa Google-ovim poslužiteljem. *Google Cloud Messaging* (GCM) 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* (FCM) izbacuje protokol koji je sada dostupan na Android i iOS pametnim mobilnim uređajima ali i na klasičnim web aplikacijama.

2.3.1. 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



Slika 2.9: Princip rada Firebase Cloud Messaging usluge

poslužiteljem kako je prikazano na slici 2.9 (1). Na Android uređajima svaka aplikacija mora imati proces koji će inicirati vezu sa FCM poslužiteljem i dobiti token jedinstven za svaku aplikaciju (2). Nakon što klijent dobije token on ga šalje svom poslužitelju koji bi trebao token spremiti ukoliko planira slati dojave tom klijentu (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.

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

Slika 2.10: Primjer JSON objekta kojeg poslužitelj šalje FCM poslužitelju za slanje dojava

2.3.2. 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 protokola HTTP ili XMML. JSON objekt se sastoji od dva atributa: `to` ili `condition`, `notification` i `data` (4). Zatim FCM poslužitelj obrađuje dojavu, sprema u svoju bazu podataka ako klijent nije spojen ili šalje dojavu klijentu (5) sa JSON objektom, kao što je prikazano na slici 2.10, 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 dojava 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 dojava.

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

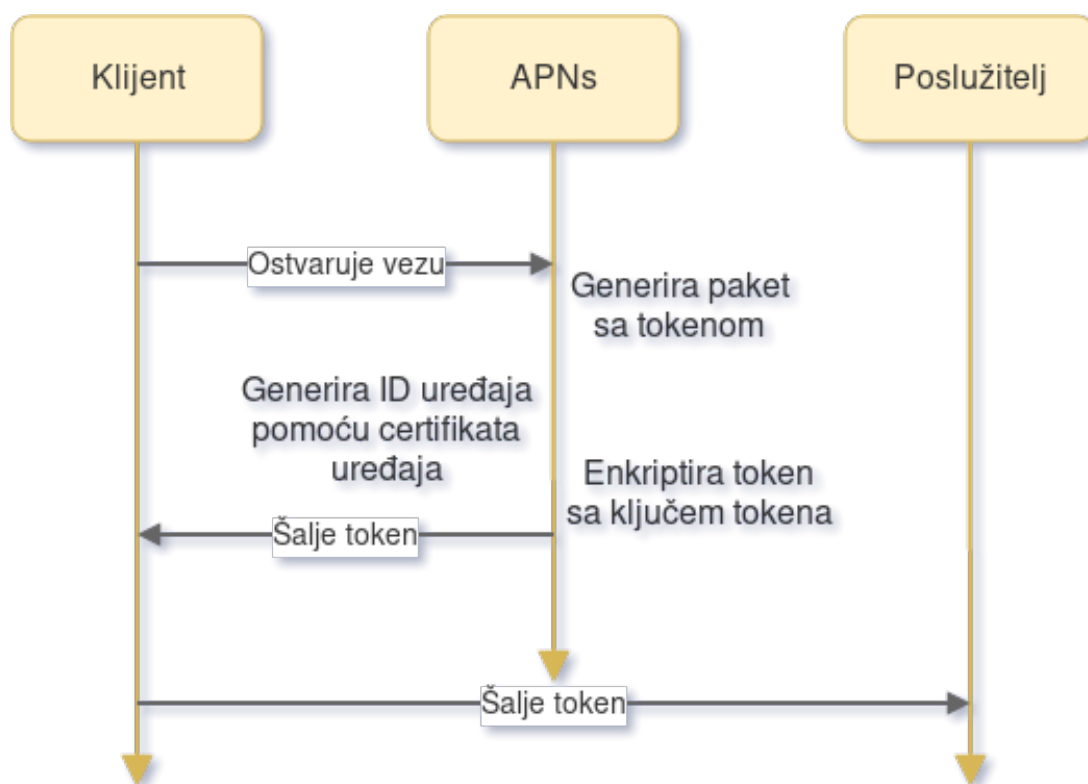
će Android aplikacija obraditi podatke i kad je u pozadini. U slučaju ako se šalju i `notification` i `data` atribut dojava će se prikazati i ako je aplikacija u pozadini dok će se `data` atribut obraditi tek kada aplikacija dođe u prednji plan.(fcm, 2017)

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 vlastite usluge za poslužiteljske dojave.

2008 godine napravljen je sustav nazvan *Apple Push Notification Service* (APNs). 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.



Slika 2.11: Upravljanje tokenom uređaja koristeći APNs

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

Slika 2.12: JSON objekt za „tihu“ dojavu

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 2.11).

2.4.2. Slanje dojava i podataka

Poslužitelj pri slanju dojava 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 attribute `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 2.12). Tiha dojava označava aplikaciji da ima dostupne dojava i da ih može dohvatiti kada to poželi.(apn, 2017)

3. Arhitektura sustava za praćenje događaja

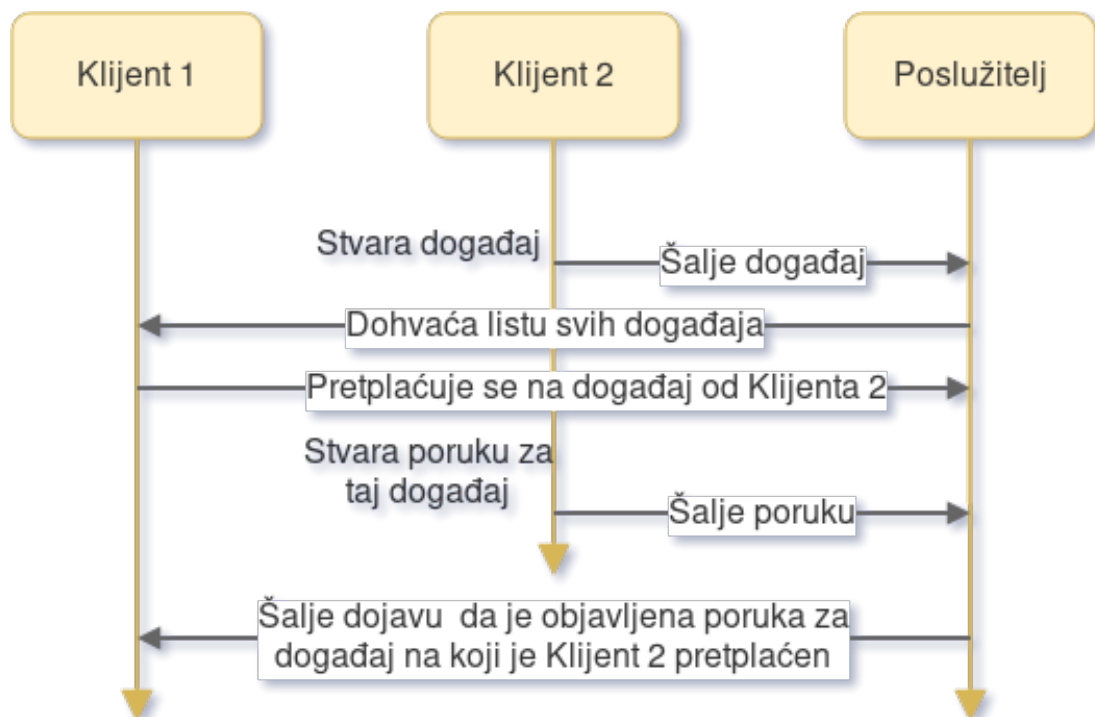
Ideja je napraviti aplikaciju koja korisniku omogućava praćenje događaja poput koncerata, rođendana i sl. i informacija vezanih uz taj događaj. Isto tako korisnici trebaju imati mogućnost stvaranja događaja za koje mogu definirati nekakve attribute poput naziva događaja, opisa, datuma i sl. Svi pratioci 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.

3.1. Protokol za razmjenu podataka

Za funkcioniranje aplikacije mora se implementirati protokol za razmjenu podataka između poslužitelja i klijenta. Za implementaciju ove aplikacije potrebno je omogućiti dvosmjernu komunikaciju između klijenta i poslužitelja.

Prilikom neke akcije, poput registracije, prijave, stvaranja novog događaja, pretplate na događaj, itd., klijent će podatke slati na poslužitelj koji će obaviti provjeru ispravnosti tih podataka, obraditi ih i odgovoriti klijentu da li su poslani podatci uspješno poslani i poslati će korisniku resurse, poput profila korisnika, liste događaja, itd., ukoliko ih je klijent zatražio.

Korisnik koji je stvorio događaj može ih i promijeniti te poslati poruke vezane uz taj događaj. Kod tih događaja potrebno je obavijestiti druge korisnike koji su se pretplatili na taj događaj o obavljenim promjenama ili novoj poruci sa poslužiteljskom dojavom kako je prikazano na slici 3.1. Ukoliko korisnik pretplatnik nije trenutno dostupan dojavu treba spremati te poslati čim postane dostupan.



Slika 3.1: Slanje događaja i pretplata na događaj

3.2. 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.

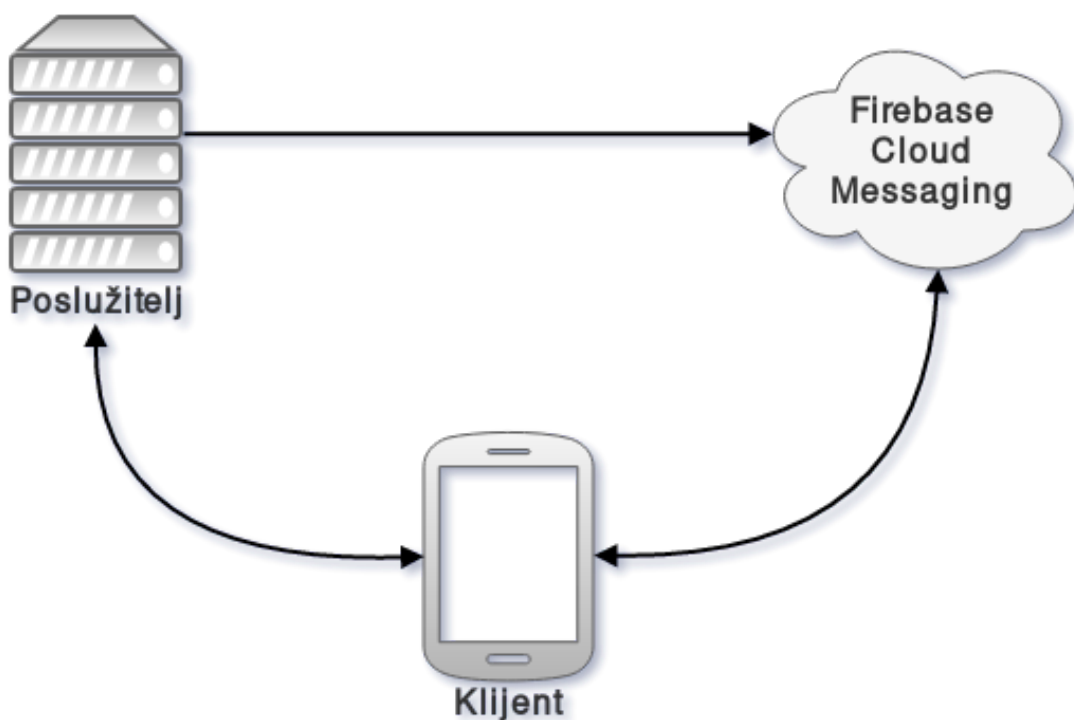
Ako se radi o ispravnim podacima klijent će podatke za prijavu privremeno spremiti na klijentski uređaj. Svaki put kada klijent želi stvoriti novi događaj, poslati poruku, pretplatiti se na događaj i sl. koristiti će te podatke kako bi poslužitelj mogao utvrditi da tu akciju obavlja korisnik koji je za to i autoriziran.

3.3. 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).

3.4. Poslužiteljske dojave

Poslužiteljske dojave se ovdje koriste kao način na koji se obavještuje 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).



Slika 3.2: Prikaz arhitekture sustava za praćenje događaja

4. Programsko ostvarenje sustava za praćenje događaja

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 dojava. 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 uslugu za poslužiteljske dojave. 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 usluga će automatski spremiti dojavu i poslati je čim korisnik ponovo ostvari vezu sa FCM poslužiteljem. Velika prednost ove usluge je ta da je više-platformska. Ovaj servis podržava Android i iOS operativne sustave ali i klasične web aplikacije.

Prije započinjanja implementacije FCM servisa na vlastitom poslužitelju sa službene stranice se mora dobiti ključ pomoću kojeg će FCM moći raspoznati našu aplikaciju od drugih. Taj ključ će se slati, zajedno sa ostalim informacijama za dojavu, FCM poslužitelju svaki put kada poslužitelj poželi poslati dojavu klijentu. Poslužitelju se mora omogućiti da se poveže sa FCM poslužiteljima pomoću HTTP veze i slanje ispravnih zahtjeva za slanje dojava.

4.2. Klijent

Pri implementaciji klijenta korištena je Android platforma. Prednost naspram klasičnih web aplikacija je ta da Android 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.

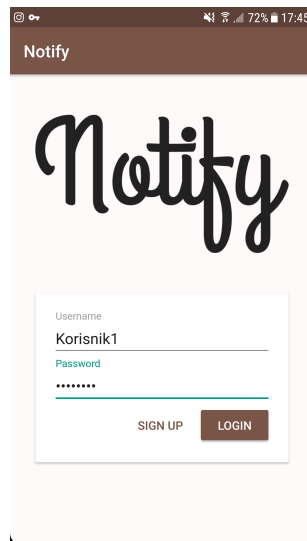
Da bi omogućili klijentu vezu sa Firebase Cloud Messaging poslužiteljem moramo koristiti Android knjižnicu za FCM pomoću koje sam napravio servis koji će biti obaviješten kada korisnik primi dojavu. Taj servis će pri primljenoj dojavi, tu dojavu obraditi i na kraju obavijestiti korisnika sa obavijesti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

4.2.1. Prijava korisnika

Pri ulasku u aplikaciju korisnika će dočekati ekran za prijavu prikazan na slici 4.1. Korisnik ima mogućnost upisati svoje podatke za prijavu, tj. korisničko ime (engl. *username*) i lozinku (engl. *password*). Nakon unosa podataka korisnik će potvrditi svoje podatke pritiskom na gumb za prijavu (engl. *login*). Prvo će aplikacija provjeriti ispravnost upisanih podataka, a zatim će ih poslati poslužitelju koji će provjeriti da li postoji upisani korisnik sa upisanom lozinkom.

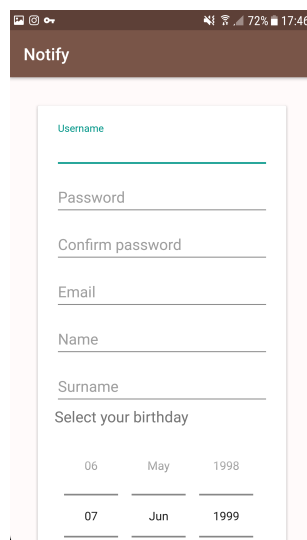
4.2.2. Registracija korisnika

Ukoliko korisnik nije prije izradio račun isti može napraviti pomoću ekrana za registraciju prikazanog na slici 4.2. Do njega može doći pritiskom na gumb za registraciju (engl. *sign up*) na ekranu za prijavu. Prilikom registracije od korisnika se traži da upiše željeno korisničko ime, lozinku, e-mail, ime, prezime i datum rođenja. Nakon

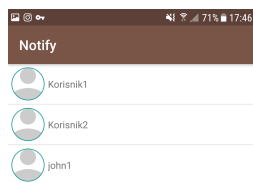


Slika 4.1: Prozor za prijavu

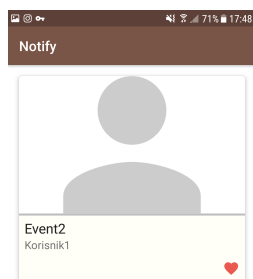
što je korisnik siguran da je upisao ispravne podatke to potvrđuje pritiskom na gumb za registraciju na dnu ekrana. Nakon pritiska aplikacija prvo provjeri podatke prije nego što ih pošalje poslužitelju. Kako se ne bi poslužitelj bespotrebno opterećivao podatke se prvo provjerava lokalno. Korisničko ime se mora sastojati od minimalno 3 do 30 alfanumeričkih znakova dok se za lozinku provjerava da li se sastoji od minimalno 8 znakova i da li sadrži barem jedno malo slovo, jedno veliko slovo i jedan broj. Nakon uspješne provjere podatke se šalje poslužitelju a kada se registracija završi korisnik je i automatski prijavljen.



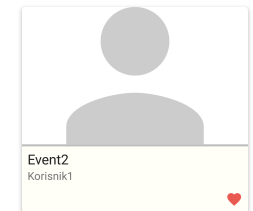
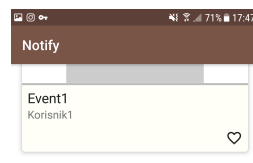
Slika 4.2: Prozor za registraciju



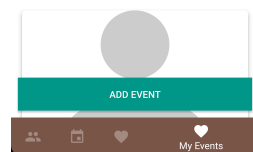
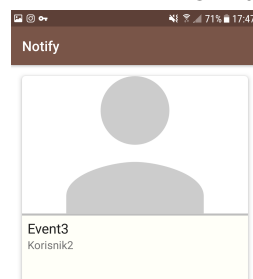
(a) Lista svih korisnika



(c) Lista praćenih događaja



(b) Lista svih događaja



(d) Lista vlastitih događaja

Slika 4.3: Glavni prozor

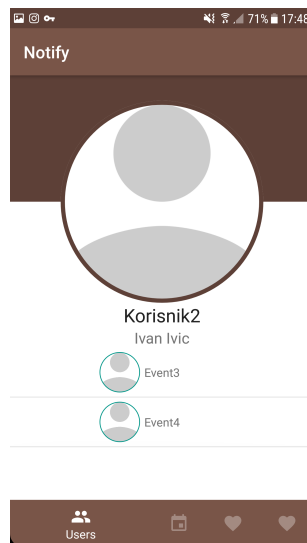
4.2.3. Glavni prozor

Nakon uspješne prijave ili registracije korisnika se prebacuje na glavni prozor koji će prikazivati listu svih registriranih korisnika prikazanu na slici 4.3a. Na dnu glavnog prozora se nalazi navigacijska traka. Na navigacijskoj traci korisnik može odabrati četiri sadržaja glavnog prozora: listu svih korisnika, listu svih događaja, listu događaja na koje je pretplaćen i listu događaja koje je stvorio.

Ako je korisnik odabrao prikaz liste registriranih korisnika tada će aplikacija poslati zahtjev poslužitelju a poslužitelj mu šalje listu svih korisnika. Aplikacija zatim korisniku prikazuje listu svih korisnika u obliku liste na kojoj su prikazani slika profila korisnika i korisničko ime.

Pritiskom na jednog korisnika, tj. jedan red otvara se ekran sa profilom tog korisnika prikazan na slici 4.4. Na tom ekranu korisnik može vidjeti osobne podatke tog korisnika poput imena, prezimena i e-mail adrese te listu događaja koje je ta osoba

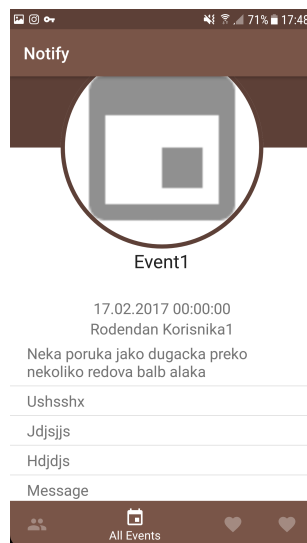
stvorila.



Slika 4.4: Prozor za prikaz profila korisnika

Korisnik također može pregledati listu svih događaja prikazanu na slici 4.3b pritiskom na drugu ikonicu na navigacijskog traci. Nakon pritiska tipke aplikacija zatraži događaje od poslužitelja te ih prikaže na ekranu u obliku liste. Svaki događaj je prikazan pomoću jedne kartice na kojoj je prikazana slika, ime događaja i korisničko ime osobe koja je stvorila taj događaj. Ako prikazani događaj nije stvorio trenutno prijavljeni korisnik također u doljnjem lijevom kutu se nalazi ikonica u obliku srca koja služi za pretplatu na taj događaj. Ukoliko je srce popunjeno crvenom bojom to znači da je korisnik trenutno pretplaćen na taj događaj, a ako je prazno znači da nije. Prilikom pritiska na ikonicu šalje se zahtjev poslužitelju za odjavu pretplate ili pretplatu na taj događaj zavisno od prijašnjeg stanja. Nakon što poslužitelj dojavu da se korisnik uspješno pretplatio ili odjavio pretplatu ikonica mijenja izgled u odgovarajući oblik. Također aplikacija javlja FCM poslužitelju da se prijavila na taj događaj tako da mu šalje jedinstveni identifikator tog događaja.

Ukoliko korisnik želi vidjeti više informacija o događaju može kliknuti na jednu od kartica i tako otvoriti ekran za prikaz profila događaja prikazan na slici 4.5. Na tom ekranu mogu se vidjeti sve informacije o događaju i sve poruke koje su poslane vezane uz taj događaj. Ukoliko je taj događaj ujedno i događaj kojeg je stvorio trenutno prijavljeni korisnik na dnu ekrana će se nalaziti gumb za dodavanje novih poruka. Pritiskom na taj gumb korisniku se prikazuje iskočni prozor sa formom za unos nove poruke. Nakon što korisnik završi sa unosom poruke i pritisne gumb za slanje poruke, ta poruka se šalje poslužitelju.

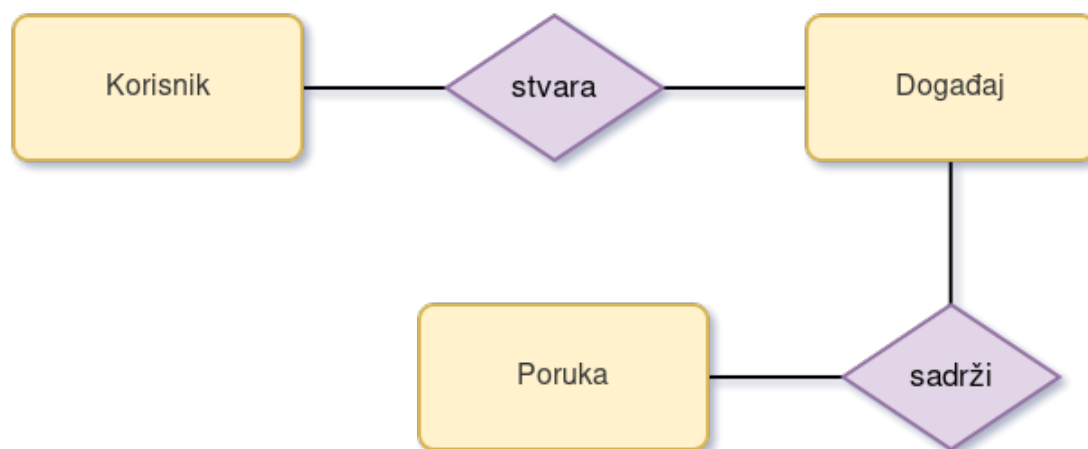


Slika 4.5: Prozor za prikaz profila događaja

Liste za pregled pretplaćenih događaja i vlastitih događaja funkcioniraju na sličan način kako i funkcionira ekran za pregled liste svih događaja s razlikom da na ekranu za listu vlastitih događaja imamo gumb za dodavanje novih događaja. Pritiskom na taj gumb korisniku se prikazuje iskočni prozor sa formom za unos novog događaja.

U svakom trenutku, čak i kada je aplikacija ugašena, korisnik može primiti poslužiteljsku dojavu ukoliko je poslana poruka vezana uz jedan od događaja na koji je trenutno prijavljeni korisnik pretplaćen. Naslov dojave će biti ime tog događaja a sadržaj dojave će biti sadržaj poruke.

4.3. Poslužitelj



Slika 4.6: ER diagram poslužiteljske baze podataka

Tablica 4.1: Popis podataka u bazi podataka

Podatak	Atributi	Opis
Korisnik	Id	Cjelobrojni broj
	FCM token	FCM token do 200 znakova
	Korisničko ime	Tekst do 50 znakova
	Lozinka	SHA1 hash od 40 znakova(160 bitova)
	E-mail adresa	Tekst do 254 znakova
	Ime	Tekst do 50 znakova
	Prezime	Tekst do 50 znakova
Događaj	Datum rođenja	Datum
	Id	Jedinstveni cijelobrojni broj
	Naslov	Tekst do 50 znakova
	Opis	Tekst do 1024 znakova
	Datum održavanja događaja	Datum i vrijeme
	Datum dodavanja događaja	Datum i vrijeme
	Kreator	Id korisnika koji je kreirao događaj
Poruka	Id	Jedinstveni cijelobrojni broj
	Sadržaj	Tekst do 1024 znakova
	Događaj	Id događaja na koji je poruka poslana
Kreatori	Korisnik	Id korisnika kreatora
	Događaj	Id događaja koji je korisnik stvorio
Pretpлатnici	Korisnik	Id korisnika pretplatnika
	Događaj	Id događaja na koji je pretplaćen

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 stvaranje SQL 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.

U bazi podataka nalaze se podatci prikazani na slici 4.6 sa svim svojim atributima prikazanim u tablici 4.1. Ti podatci su stvoreni od strane korisnika i njima se prikazuju. Pomoću tih podataka korisnici mogu odlučiti na koje događaje se žele pretplatiti.

```
POST /events/5/messages HTTP/1.1
Authorization: Basic TWFuIGlzIGRpc3Rpbmd1aX=

{
    "fcm_token": "xHSnx2421...",
    "username": "Example1",
    "password": "2fd4e1c67a..."
    ...
}
```

Slika 4.7: Primjer zahtjeva za registraciju

Prilikom interakcije sa klijentom bitno je da poslužitelj može pouzdano utvrditi identitet korisnika koji šalje upite. Fokus ovog rada nije na sigurnosti pa je stoga korišten tehnika *HTTP Basic authentication* (BA). Sustav od korisnika zahtjeva da uz svaki zahtjev za koji je potrebna autentikacija šalje zaglavlje u obliku `Authorization: Basic {Base64 šifra}` gdje je `{Base64 šifra}` korisničko ime i lozinka šifrirana sa *Base64* algoritmom. Primjer korištenja ovakve autentikacije možemo vidjeti na slici 4.7. Sve akcije iz tablice 4.2 koje koriste HTTP metode `POST` ili `PATCH` te metode za dohvat liste vlastitih događaja i liste pretplaćenih događaja zahtjevaju od korisnika da sa zahtjevom pošalje zaglavlje `Authorization`.

Tablica 4.2: Popis akcija na poslužitelju

Metoda	Putanja	Opis
POST	/users/login	Prijava korisnika
POST	/users	Registracija korisnika
GET	/users	Dohvat liste svih korisnika
GET	/users/{id}	Dohvat profila korisnika pomoću id-a
PATCH	/users/{id}/token	Osvježavanje FCM tokena
POST	/events	Stvara događaj
GET	/events	Dohvat liste svih događaja
GET	/events/{id}/events	Dohvat liste događaja
GET	/users/{id}/followed	Dohvat profila korisnika pomoću id-a
PATCH	/users/{userId}/follow/{eventId}"	Pretplaćivanje na događaja
PATCH	/users/{userId}/unfollow/{eventId}"	Prekid pretplate na događaj
POST	/events/Id/messages	Stvara poruku za zadani događaj

5. Zaključak

U ovom završnom radu predstavljena su tehnologije za poslužiteljske dojave, od starijih rješenja koja su imitirala poslužiteljske dojave poput *long polling* tehnike do novih tehnologija poput protokola *WebSocket* i usluga poput usluge *Firebase Cloud Messaging*. 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 uslugom *Firebase Cloud Messaging* sam se imao prilike više upoznati kroz praktični rad gdje sam paradigmu poslužiteljskih dojava koristio za dojavljivanje 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.

WHATWG. Html standard, lipnja 2017. URL <https://html.spec.whatwg.org/#server-sent-events>. 8. lipnja 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: poslužiteljske dojave, web-usluga, Comet, Server Sent Events, WebSocket, Firebase Cloud Messaging, Apple Push Notification Service

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: push notifications, web-service, Comet, Server Sent Events, WebSocket, Firebase Cloud Messaging, Apple Push Notification Service