Univerzitet u Beogradu
Matematički fakultet

Petar Rutešić

Uporedni pregled i analiza tehnika povezivanja mobilnih klijentskih aplikacija i servera

Master rad

Beograd

2017.

Univerzitet u Beogradu - Matematički fakultet

Master rad

Autor:	Petar Rutešić
Naslov:	Uporedni pregled i analiza tehnika povezivanja mobilnih klijentskih aplikacija i servera
Mentor:	prof. dr Saša Malkov
Članovi komisije:	prof. dr Miodrag Živković doc. dr Milena Vujošević-Janičić
Datum:	

Apstrakt

U radu su opisane tehnike povezivanja mobilnih aplikacija i servera. Za svaku od tehnika je opisana arhitektura i konkretna tehnologija koja se koristi.

Na početku je obrađeno povezivanje mobilnih aplikacija i servera korišćenjem veb servisa. Dat je kratak pregled razvoja veb servisa i njihove vrste i karakteristike, sa akcentom na *REST*-servis (engl. *REpresentational State Transfer service*). Nakon veb servisa, obrađeni su soketi kao jedan od mogućih načina povezivanja. Glavni akcenat je na protokolu *WebSocket*. Nadalje su obrađena potiskivana obavještenja. Dat je opis konkretnog servisa za slanje potiskivanih obavještenja i primjeri poruka koje se razmijenjuju prilikom komunikacije.

Pošto je akcenat na mobilnoj platformi Android, obrađene su i osnovne karakteristike operativnog sistema Android.

Na kraju, opisana je arhitektura konkretne Android aplikacije koja koristi sve u radu navedene tehnike i tehnologije. Implementacija aplikacije je opisana kroz reprezentativne slučajeve korišćenja, nakon čega je data diskusija o odabiru konkretne tehnike povezivanja za dati slučaj korišćenja. Za odabranu tehniku su dati primjeri poruka koje se razmijenjuju između mobilne aplikacije i servera.

Sadržaj

1 Uvod

Razvoj veba je donio pogodnosti koje su prethodnim generacijama mogle djelovati kao naučna fantastika – u komforu sopstvenog doma moguće je pristupiti ogromonoj biblioteci ljudskog znanja i kulture. Razvojem mobilnih tehnologija korisnici više nisu ograničeni na dom ili kancelariju – pristup vebu i njegovim servisima imaju sa bilo koje tačke na planeti koja je pokrivena internetom. Međutim, veb se nije zaustavio samo na tome da bude kolekcija članaka i prezentacija, već se danas koristi kao pokretač mnogih aplikacija. Činjenica je da većina modernih aplikacija za mobilne platforme razmijenjuje određenu količinu podataka sa serverima. Današnji pametni telefoni i ostali mobilni uređaji zahvalnost za svoju popularnost donekle duguju i vebu, koji je postao osnova komunikacije između aplikacija za mobilne platforme i servera. U principu, aplikacije za mobilne platforme su se samo prilagodile postojećim konceptima veba, što je prilično olakšavajuća činjenica.

Veb servisi, među kojima prednjače *REST*-oliki, su omogućili da klijentski i serverski dio aplikacije komuniciraju korišćenjem postojeće veb infrastrukture, preciznije protokola *HTTP* (engl. *HyperText Transfer Protocol*). Zahvaljujući osobinama veba, klijentski i serverski dio aplikacije mogu biti implementirani u različitim programskim jezicima, na različitim platformama. Ipak, sa stanovišta komunikacije u realnom vremenu, performansi i potrošnje resursa (baterije, mreže) koji su veoma bitni za mobilne uređaje *HTTP* ne predstavlja optimalno riješenje. Stoga su razvijeni mnogi protokoli i servisi koji uglavnom dobro uklanjaju nedostatke (ukoliko se uopšte može govoriti o nedostacima s obzirom na to da je *HTTP* danas postao mnogo više od onoga za šta je prvobitno predviđen). Neki od tih protokola i servisa su servis za slanje potiskivanih obavještenja (engl. *push notifications*) i *WebSocket*-i, koji se u nekim segmentima naslanjaju na *HTTP*.

Postoji nekoliko operativnih sistema za mobilne uređaje, među kojima su najpopularniji: iOS, Android, *Windows Mobile*, *BlackBerry OS* itd. Najzastupljeniji operativni sistem današnjice je Android. Njegov razvoj je rezultat rada komanije Gugl (engl. *Google)*.

Na početku rada će biti izložene karakteristike veb servisa kao osnovne tehnike povezivanja mobilne aplikacije sa serverom, sa posebnim akcentom na *REST*-olike veb servise. U nastavku rada su prezentovani osnovni koncepti koji stoje iza *WebSocket*-a i potiskivanih obavještenja. Nakon toga slijedi implementacija aplikacije za operativni sistem Android. Aplikacija komunicira sa aplikacionim serverom koristeći sve tri navedene tehnike. Cilj je da se kroz aplikaciju na praktičnim primjerima pokaže u kojim slučajevima se navedene tehnike mogu optimalno koristiti. Aplikacija je razvijena u razvojnom okruženju Android Studio 2.2. Za realizaciju veb servisa je korišćen *Laravel*, dok je *WebSocket* server (*RatchetPHP*) implementiran u *PHP*-u. Za komunikaciju putem potiskivanih obavještenja je korišćen *Firebase Cloud Messaging* (*FCM*), kompanije Gugl. Razvijena je Android aplikacija KlikTaxi koja služi da omogući korisnicima da naruče taksi vozilo u direktnoj komunikaciji sa dispečerima različitih taksi udruženja.

2 Veb servisi

Definicija veb servisa, data od strane W3C konzorcijuma glasi: “Veb servis je softverski sistem dizajniran da podrži interoperabilnu interakciju između mašina putem mreže” [1]. Unutar definicije se ponekad pojam mašina zamijenjuje sa pojmom aplikacija [2].

W3C (engl*. World Wide Web Consortium)* je internacionalna organizacija čije organizacije članice, kao i zaposleni i javnost rade zajedno na razvoju veb standarda. Organizaciju vodi Tim Berners Li, otac onog što je danas poznato kao *World Wide Web,* tj. svjetska mreža ili skraćeno - veb.

Realizacija veb servisa se najčešće vrši putem *HTTP* protokola, mada mogu biti korišćeni i drugi protokoli. *HTTP* poruka koja dolazi od klijenta ka serveru se naziva zahtijev (engl. *request*). *HTTP* poruka koja dolazi od servera ka klijentu naziva se odgovor (engl. *response).* O samom *HTTP* protokolu će biti više riječi u nastavku.

Veb koji se danas koristi obiluje servisima: pretraživači veba, aukcijski sajtovi, socijalne mreže itd. Umjesto instaliranja svih ovih programa i podataka na uređaj, dovoljno je instalirati jedan program – pregledač veba. Putem njega su dostupni svi servisi i podaci.

Korisnici pristupaju određenim servisima na vebu tako što u svoj pregledač veba unesu adresu do servisa (npr. adresu pretraživača veba www.google.com). Kao odgovor korisnik dobija *HTTP* poruku koja sadrži uputstva kako će taj odgovor biti prikazan korisniku, gotovo uvijek u vidu *HTML* (engl. *HyperText Markup Language*) dokumenta. Dalje, čovjek može unijeti neke ključne riječi po kojima će vršiti pretragu i poslati novi *HTTP* zahtijev sa tim ključnim riječima ka servisu. Nakon toga, od servisa dobija nazad *HTTP* poruku sa odgovorom, koji se sastoji od spiska relevantnih rezultata, hiperlinkova ka tim rezultatima i slično, opet u vidu *HTML* datoteke.

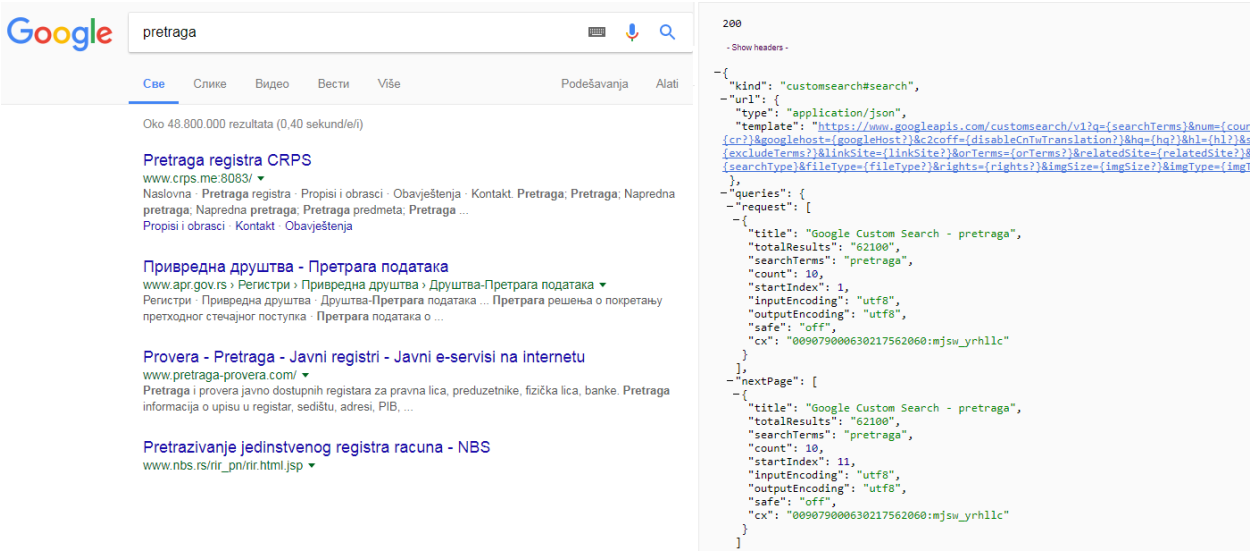
Prethodni primjer je slučaj kada čovjek pristupa servisu, a servis vraća rezultat koji je razumljiv čovjeku. U slučaju da umjesto čovjeka aplikacija šalje zahtijev servisu, javlja se problem interpretacije odgovora.

Kada čovjek pregleda odgovor koji je dobio od veb servisa, on vizuelno tumači reprezentaciju odgovora (npr. veb stranu rezultata pretrage) i uočava šta mu je bitno a šta ne. Takvo tumačenje odgovora bi za mašinu/aplikaciju bilo previše kompleksno. Zbog toga veb servis isporučuje rogobatni *XML* (engl. *eXstensible Markup Language*), *JSON* (engl. *JavaScript Object Notation*) ili neki drugi format odgovora koji mašina može lako da tumači. Sada aplikacija može da koristi tako dobijeni odgovor kao ulazne podatke nad kojima vrši određene operacije.

U prethodno navedenim primjerima je i osnovna razlika između veb aplikacije i veb servisa.

- Veb aplikacije najčešće sa odgovorom šalju i *HTML* sa opisom kako će taj odgovor biti prikazan (primjer kada čovjek pretražuje internet);
- Veb servis kao odgovor šalje samo potrebne podatke u određenom formatu (*XML*, *JSON*, binarni format itd.);

Na slici 1 su dati primjeri odgovora servisa u oba slučaja. Na lijevoj slici pregledač veba samo prikazuje odgovor koji je dobio od strane servisa, a čovjek ga tumači. Na slici desno klijent (aplikacija) mora da protumači *JSON* odgovor i iz njega pročita potrebne podatke (npr. ukupan broj rezultata pretrage).



Slika 1 - Dva tipa odgovora veb servisa

2.1 Nastanak veb servisa

Veb servisi su nastali od poziva udaljenih procedura *RPC* (engl. *Remote Procedure Call*), mehanizma u distribuiranom računarskom okruženju *DCE* (engl. *Distributed Computing Environment*) početkom 90-ih godina 20. vijeka. *DCE* je razvojni okvir nastao na *UNIX* sistemu. Majkrosoft (engl. *Microsoft*) je brzo napravio svoju verziju poznatu kao *MSRPC*, koja je služila za komunikaciju između procesa na operativnom sistemu Vindouz (engl. *Windows*) [3].

Krajem 90-ih godina 20. vijeka dolazi do razvoja *XML-RPC* tehnologije, koja se može smatrati praocem veb servisa u današnjem smislu. Radi se veoma jednostavnom sistemu koji podržava osnovne tipove podataka. Zasnovan je na *HTTP*-u i za razliku od *DCE-RPC* gdje su poruke binarne, ovdje su u *XML* (tekstualnom) formatu. Prevođenje podataka iz memorije u tekstualni format (engl. *marshaling*) i obrnut proces (engl. *unmarshaling*), kao i oslanjanje na *HTTP* i *SMTP* (engl. *Simple Mail Transfer Protocol*) protkol je izuzetno pojednostavilo stvari u distribuiranom programiranju [3]. Protokol za razmijenu je javan i standardizovan, poruke su lako čitljive i lako se mogu prevesti u odgovarajuću strukturu u programskom jeziku, što je u mnogome olakšalo komunikaciju između čvorova koji su implementirani u različitim programskim jezicima, često i jezicima različitih paradigmi. Sve što je potrebno je da

programski jezici posjeduju biblioteke za *HTTP* komunikaciju, kao i biblioteke za čitanje tekstualnog formata poruke.

2.2 Tipovi veb servisa

Razlikuju se dva najčešća tipa veb servisa:

- Veliki veb servisi (engl. *Big web services*) – poznati još kao *SOAP* bazirani servisi;
- *REST*-oliki (engl. *RESTful*) veb servisi.

SOAP (engl.*Simple Object Access* Protocol) predstavlja komunikacioni protokol, a ne vid arhitekture. Prenos podataka između klijenta i servera se vrši korišćenjem *XML*-a. *SOAP* protokol omogućava komunikaciju izmedju aplikacija na različitim operativnim sistemima i različitim tehnologijama tako što aplikacije razmenjuju poruke "dogovorenog" formata. Preduslov je da bez obzira na različitosti svi mogu koristiti *HTTP* protkol. Kod ovakve vrste servisa, klijent šalje *SOAP* zahtijev/poruku servisu, a kao odgovor od servisa takođe dobija *SOAP* poruku.

Opis veb servisa je dat *WSD*-om (engl. *Web Service Description).* Kao jezik za opis se koristi *WSDL* (engl. *Web Service Description Language*) predstavlja *XML* dokument sa ekstenzijom **.wsdl**.

Radi pojednostavljenja, slijedi primjer kalkulatora. Neka klijent šalje dva broja veb servisu i želi da vrši aritmetičke operacije nad njima. Prvo treba da preuzme *WSDL* opis servisa, da bi znao koje procedure klijent može pozivati, koje parametre one uzimaju, koje vraćaju kao odgovor itd. Nakon preuzimanja *WSDL* datoteke klijent vidi da postoje procedure saberi(), oduzmi(), pomnozi(). Svaka od njih uzima dva broja kao argumente, a kao rezultat vraća jedan broj. Klijent sada može da pošalje *SOAP* poruku koja sadrži ta dva broja, a kao rezultat dobije *SOAP* poruku u kojoj je rezultat aritmetičke operacije nad dva broja.

REST (engl. *REpresentational State Transfer*) je definisan u doktorskoj disertaciji Roja Fildinga. *REST* je stil ahritekture, dok je *SOAP* protokol. *REST* nije standard sam po sebi, ali *REST*-olike implementacije koriste standarde, kao sto su *URI*, *HTTP*, *JSON* i *XML* [4]. Da bi pojam *REST*-a bio razumljiv, potrebno je prvo razumjeti šta znači *CRUD*. *CRUD* je skraćenica od piši (engl. *Create*), čitaj (engl. *Read*), izmijeni (engl. *Update*) i obriši (engl. *Delete*). *CRUD* predstavlja spisak operacija koje je moguće raditi nad nekim resursom.

Kod ovog tipa servisa, resursi (npr. statičke stranice, datoteke, podaci iz baze itd.) imaju sopstveni uniformni identifikator resursa koji ih identifikuje, skraćeno *URI* (engl. *Uniform resource identifier*). Pristup resursima je definisan *HTTP* protokolom, a svaki *HTTP* zahtijev je vrsta operacije (koja piše, čita, mijenja ili briše podatke). Isti *URI* se koristi za sve operacije nad resursom ali se mijenja *HTTP* metod koji definiše vrstu operacije. *REST* koristi *HTTP* metode: *GET*, *POST*, *PUT*, *DELETE* – ovdje je primjetna analogija sa CRUD-om.

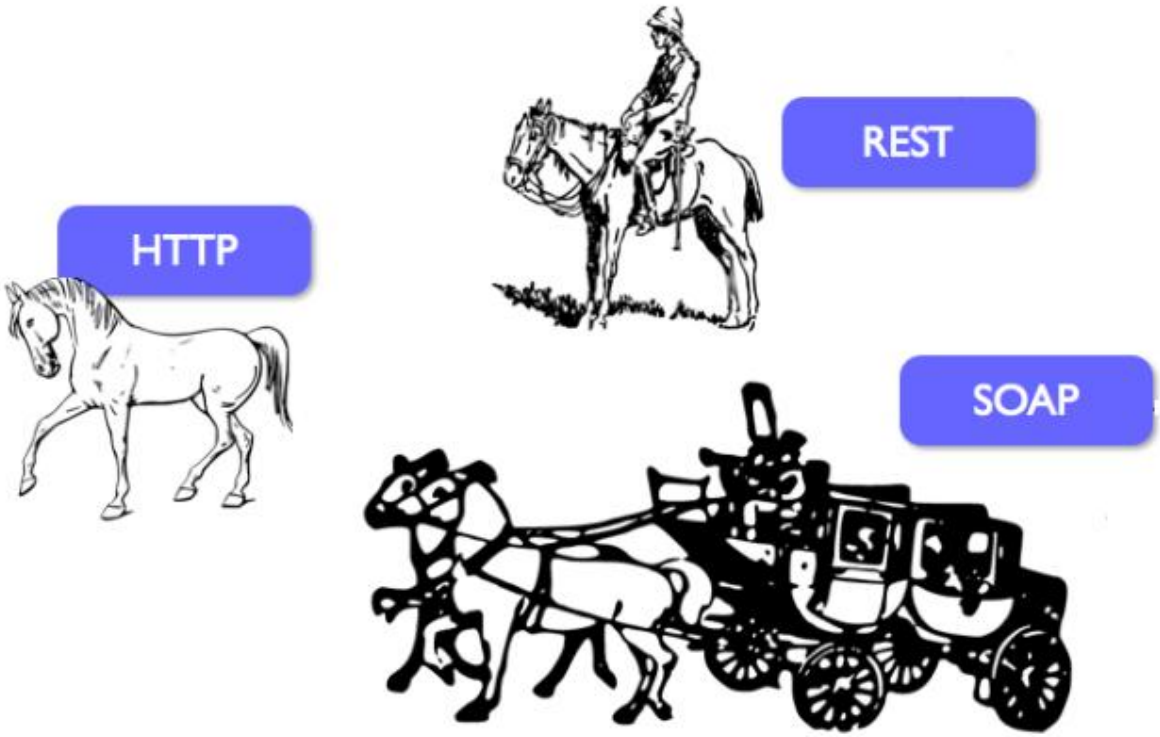
Kao odgovor veb servisa najčešće se dobijaju podaci u *JSON* formatu, mada je moguće i u drugim formatima, npr. *XML*.

Danas su se *REST*-oliki servisi izdvojili kao dominantni servisi, čime su potisnuli *SOAP* i WSDL u drugi plan jer su značajno jednostavniji za upotrebu. Zasnivaju se na *REST* arhitekturi, veoma su fleksibilni i jednostavni za razumijevanje.

REST arhitektura se sastoji od klijenta i servera. Klijent šalje zahtijev serveru, server procesira zahtijev i vraća odgovor klijentu. *REST*-oliki servisi se koriste:

- kod operacija koje ne koriste stanja (ukoliko neka operacija treba da bude nastavljena onda *REST* nije pravi pristup i *SOAP* je potencijalno bolje rešenje);
- kod situacija gde je moguće keširanje (ukoliko informacija može biti keširana zbog operacija koje ne koriste stanja onda je ovaj pristup odličan);

Na slici 2 je ilustrovana razlika između *SOAP* i *REST*-olikih veb servisa. Konj predstavlja *HTTP* protokol. Kod *REST*-olikih veb servisa, sve što je potrebno za transfer podataka je konj (*HTTP* protokol), dakle prenos podataka se direktno oslanja na njega. Kod *SOAP*-a je drugačija situacija, gde su podaci umotani u *SOAP* omotač (kočija) koju vuče *HTTP* (konj).



Slika 2 - Odnos *SOAP* i *REST* servisa

2.3 Podjela veb servisa prema arhitekturi

Razlikujemo tri arhitekture veb servisa: *REST*-olika, arhitektura orijentisana na resurse, *RPC* i *REST RPC* hibridna [2].

2.3.1 REST-olika, arhitektura orijentisana na resurse

Pojam *REST*-olika arhitektura se ne odnosi samo na arhitekture pogodne za veb servise, mada su za ovaj rad one najbitnije. Kada se govori o *REST*-olikim veb servisima, misli se na servise koji liče na običan veb [2]. Dobra arhitektura za *REST*-olike veb servise je ROA – resurs orijentisana arhitektura.

Kod *REST*-olikih arhitektura informacija o metodu se nalazi u *HTTP* zaglavlju (npr. *GET* ili *POST*), o čemu će više riječi biti kasnije. Kod ROA-e informacija o oblasti koja definiše o kom se resursu radi ide u *URI*-u. Na primjer, u *HTTP* zahtijevu:

GET /oprema/alati *HTTP*/1.1

prema *REST*-olikom resurs-orijentisanom veb servisu, na osnovu navedene *HTTP GET* metode može se zaključiti o kojem se metodu radi, a na osnovu *URI*-a o kojem resursu se radi.

2.3.2 RPC arhitektura

Veb servisi u stilu *RPC* (engl. *Remote procedure call*) prihvataju paket (omotač) sa podacima od strane klijenta, i omotač istog formata vraćaju nazad klijentu. Najčešći format omotača je *HTTP*. Postoji i implementacija u vidu *SOAP*-a gdje se *SOAP* omotač ugrađuje u *HTTP* omotač. Postoje dvije ključne razlike u odnosu na *REST*-olike veb servise:

- Informacije o metodu i oblasi se nalaze u omotaču umjesto u samom *HTTP* zahtijevu;
- Svaki *RPC* servis ima svoj riječnik (kao što svaki program ima raznolike nazive procedura i funkcija), za razliku od *REST*-olikih veb servisa koji koriste standardne *HTTP* metode.

Primjeri ovakvih arhitektura su veći dio *SOAP* servisa i danas gotovo potisnuti *XML-RPC*. *SOAP* je donekle i nastao iz *XML-RPC*-a.

2.3.3 Hibridni *REST RPC*

Ovakva arhitektura nije ni prava *REST* ni *RPC*. Već je navedeno da je za *REST*-olike veb servise bitno da se informacije o metodu i informacije o resursu nalaze u samom *HTTP* metodu i *URI*-u, tačnije da se nalaze u samom *HTTP* zahtijevu. Ukoliko bi se poštovali postulati *REST*-a o kojima će biti riječi kasnije, metod *HTTP POST* se ne bi

smio koristiti za čitanje podataka, već samo za pravljenje novog objekta. Međutim, moguće je da veb servis u jednom segmentu poštuje *REST* postulate i metodom *POST* pravi novi objekt, a u drugom segmentu vrši izmjenu nad resursom putem iste metode što je primjer *RPC* arhitekture. Ovakva arhitektura je rezultat razvijanja veb servisa na način na koji se razvija veb aplikacija [2].

2.4 Tehnologije

Navedene arhitekture koriste različite tehnologije. Jedna od najznačajnih tehnologija je HTTP protokol, zato što se najčešće koristi u svim navedenim arhitekturama. Tehnologije XML i JSON predstavljaju najznačajnije načine reprezentacije resursa.

2.4.1 HTTP protokol

Protkol za transfer hiperteksta *HTTP* (engl. *HyperText Transfer Protocol*) je jedan od široko rasprostanjenih aplikacionih protokola na Internetu. Protokol je relativno jednostavan [5]. Tvorac ovog protokola, Tim Berners Li, je prilikom stvaranja protokola imao cilj da protokol bude što jednostavniji, čime bi se pospiješio brzi razvoj veba. Sam protkol je prvobitno bio namijenjen za transfer *HTML* dokumenata, mada ga danas ne koriste samo pregledači veba već i veliki dio softvera koji koristi internet. Od prve verzije pa do sada je realizovano nekoliko verzija *HTTP* protokola.

Osnova ovog protokola jeste komunikacija bez postojanja stanja (engl. *stateless*) između klijenta i servera, po principu zahtijev-odgovor. Server koji se nalazi na adresi ***A*** osluškuje određeni port – najčešće port 80 ili 8080. Klijent koji je na adresi ***B*** šalje zahtijev serveru na adresi ***A*** i *HTTP* portu i čeka odgovor. Nakon što server primi zahtijev, ukoliko je moguće opslužuje ga i šalje odgovor klijentu na adresi ***B***. *HTTP* protokol za transfer poruka koristi TCP/IP protokol.

***HTTP* 0.9** je prva verzija *HTTP* protkola iz 1991. godine čiji je direktni tvorac Tim Berners Li. Protokol je toliko jednostavan da bi se komotno mogao nazvati “Jednolinijski protkol” (engl. *one line protocol*) [5]. Osnova protkola je klijent koji šalje zahtijev ka serveru - niz karaktera (engl. *string*) u jednom redu. Taj zahtijev je počinjao sa ključnom riječju *GET*, nakon čega bi bio naveden naziv dokumenta. Nakon toga, server isporučuje *HTML* dokument klijentu i konekcija se raskida. Dakle nema metapodataka - zaglavlja i slično, samo zahtijev u jednom redu i hipertekst u odgovoru.

***HTTP*/1.0** je druga verzija protkola, nastala kao rezultat istovremenog razvoja pregledača veba, *HTML*-a i internet infrastrukture. Ova verzija je nastojala da otkloni nedostatke kod prvobitne verzije, od kojih su glavni ti što je protokol zasnovan samo na hipertekst dokumentima i da ne postoje metapodaci o zahtijevu i odgovoru. Prvobitno su ti nedostaci otklanjani ad-hoc, svako je mogao da pravi svoju implementaciju protokola onako kako mu odgovara, pa bi onda tu implementaciju drugi ljudi eventualno usvajali. Formalni standard nikada nije definisan [5], mada je

1996. objavljen dokument RFC 1945 koji sumira najbolje osobine do tada eksperimentalnih implementacija.

Razlike između verzija 0.9 i 1.0 su sledeće:

- Zaglavlje zahtijeva se sastoji od više redova;
- Odgovor se takođe sastoji od više redova, gdje je prvi red statusni kod;
- Odgovor ne mora biti samo hipertekst, već može da bude i bilo koji drugi objekat – hipermedij;
- Pojava novih metoda (*PUT*, *HEAD*, *POST*, *DELETE*).

***HTTP*/1.1** je treća verzija protokola. Ovaj protokol je definisan kao IETF standard 1997. godine dokumentom RFC 2068, pola godine nakon objave verzije 10 (tj. RFC 1945). Nekoliko poboljšanja je 1999. godine ugrađeno u standard RFC 2616. Jedno od glavnih poboljšanja u odnosu na prethodne verzije jeste da više nije potrebno svaki put otvarati novu TCP konekciju prilikom slanja zahtijeva za resursima ka istom serveru, jer su uvedene veze sa živim paketima (engl. *keep-alive packets*). U specifikaciji je dodat veliki broj parametara u zaglavljima, među kojima su polja vezana za: sadržaj, jezik, set karaktera, keširanje, kolačiće itd.

***HTTP*/2** je četvrta, savremena verzija protokola. Sam protokol nije pretrpio gotovo nikakve izmjene semantike i sintakse, tako da se mogu koristiti isti *API*-ji kao i za ranije verzije protokola. Akcenat kod verzije 2 je na performansama; načinu na koji se koriste resursi mreže i servera kao i smanjivanju kašnjenja sa korisničkog aspekta. Osnova za *HTTP*/2 je protokol SPDY, razvijen prvenstveno od strane Gugla za prenos veb sadržaja.

Najznačajnije i najkorišćenije *HTTP* metode primjenjive nad resursima su:

- *GET* – koristi se za preuzimanje reprezentacije resursa koji je naveden u zahtijevu;
- *POST* – koristi se za snimanje podataka na serveru;
- *PUT* – ukoliko resurs iz *URI*-a već postoji, koristi se za izmjenu postojećih podataka, u protivnom se ponaša kao i *POST*;
- *DELETE* – koristi se za brisanje resursa.

Osim njih koriste se još i *HEAD*, *PATCH*, *OPTIONS*, *TRACE*. Primjer *HTTP* sesije (para zahtijev/odgovor) je dat na slici 3.

Osnovna struktura *HTTP* zahtijeva je sledeća:

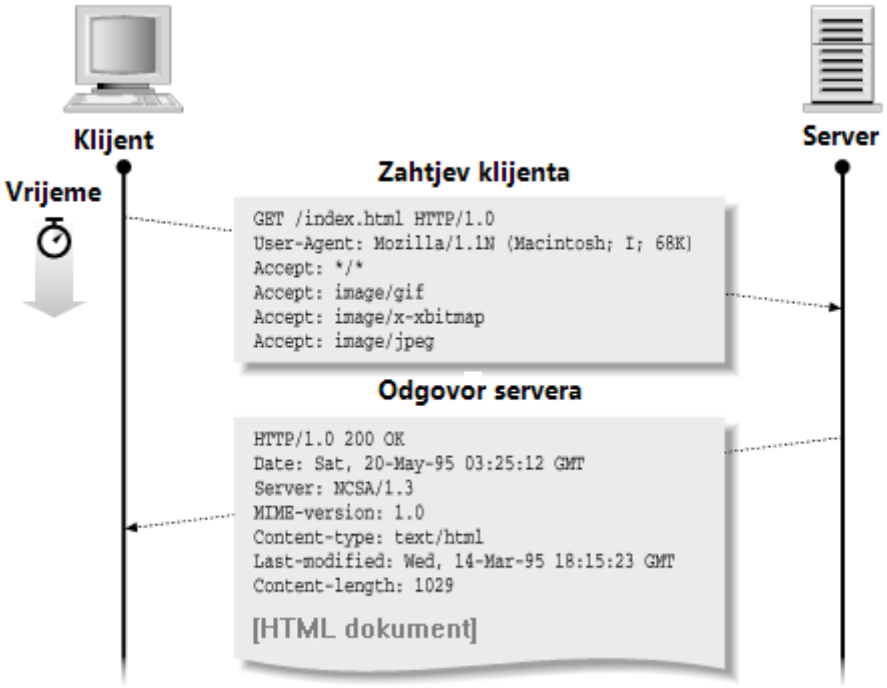
- Naziv metode (*GET*, *POST* itd.), *URI*, oznaka protkola (*HTTP*/1.0, *HTTP*/1.1)
- Zaglavlje zahtijeva u vidu parova ključ: vrijednost.

Osnovna struktura *HTTP* odgovora je sledeća:

- Oznaka protkola (*HTTP*/1.0, *HTTP*/1.1), statusni kod (npr. 200 OK) ;
- Zaglavlje odgovora u vidu parova ključ – vrijednost.

- Tijelo odgovora–u konkretnom slučaju *HTML* dokument, generalno bilo koja reprezentacija resursa koju klijent razumije.

HTTP/1.1 definiše 47 parametara zaglavlja (parova ključ – vrijednost) za zahtijeve i/ili odgovore. Dobar dio njih se rijetko kada koristi, dok se neki gotovo podrazumijevano koriste. Jedno od bitnih unapređenja kod *HTTP*/1.1 jeste pregovaranje o sadržaju putem zaglavlja (engl. *content negotiation*) [6]. U primjeru na slici 3 je zathjev klijenta u kojem on šalje parametar zaglavlja *Accept* čija je vrijednost tip sadržaja koji on razumije (u konkretnom slučaju to su bilo koji tip, slika GIF formata, slika bitmap formata i slika JPEG formata). U zaglavlju odgovora se nalazi parametar *Content-type* koje govori koji je tip resursa koji server isporučuje klijentu (u konkretnom slučaju to je *HTML* dokument, dužine 1029 bajtova).



Slika 3 - Primjer *HTTP* zahtijeva i odgovora

Statusnih kodova ima ukupno 41. Služe da informišu klijenta šta se desilo na serveru prilikom obrade zahtijeva. Predstavljaju trocifreni broj na početku *HTTP* odgovora. Mogu se svrstati u pet grupa:

- 1xx: Informativni (engl. *Informational*) – prilikom pregovorima izmedju *HTTP* klijenta i servera;
- 2xx: Uspješni (engl. *Successful*) – prilikom uspješne obrade zahtijeva na serveru;
- 3xx: Redirekcija (engl. *Redirection*) – ukoliko se zahtev koji je klijent poslao nije izvršio na serveru, ali uz određene izmjene bi mogao;
- 4xx: Greška na klijentu (engl. *Client error*) – prilikom lošeg *HTTP* zahtijeva od strane klijenta koji se ne može izvršiti na serveru;

- 5xx: Greška na serveru (engl. *Server error*) – prilikom greške pri izvršavanju na serveru.

Neki od najčešćih kodova su: 200 (*OK*), 400 loše formiran zahtijev (engl. *Bad request*), 404 nije pronađeno (engl. *Not found*), 301 premješteno (engl. *Moved permanently*), 500 interna greška servera (engl. *Internal server error*).

Jedan od glavnih problema ovog protokola je što je su podaci koji se prenose u sirovom (engl. *plain text*) obliku, što znači da su prilično osjetljivi na napade. Ovaj problem je prevaziđen primjenom SSL protokola, preciznije primjenom zaštite na nivou protokola transportnog sloja TLS (engl. *Transport Security Layer*). Podaci se šifriraju prije slanja i dešifriraju nakon prijema. Port koji koristi ovaj protokol je 443.

2.4.2 JSON

JavaSkript objektna notacija *JSON* (engl. *JavaScript Object Notation*) je jednostavan format za razmjenu podataka. Prilično lako se generiše i tumači, a i ljudima je mnogo čitljiviji od formata poput *XML*-a. Ovaj format je nezavisan od jezika ali koristi konvencije koje su poznate progamerima koji imaju iskustva u radu sa C familijom jezika (C, C#, Java, Perl, *PHP*, Pajton itd) [7]. U principu, *JSON* je u tekstualnom formatu, i tek nakon evaluacije korišćenjem parsera ta tekstualna reprezentacija se prevodi u odgovarajuću strukturu u zavisnosti od programskog jezika čiji je parser. Jedan *JSON* fajl se može sastojati od sledećih elemenata:

- Objekata - skupa parova naziv-vrijednost; u različitim jezicima, nakon evaluacije *JSON* niske, različito se realizuju ove strukture. Neke od tih struktura su: objekti, zapisi, heš tabele itd. Počinje otvorenom vitičastom zagradom „{“ , nakon čega je skup parova naziv-vrijednost razdvojenih zapetom. Naziv je odvojen od vrijednosti simbolom „:“. Na kraju je zatvorena vitičasta zagrada „}” (Primjer 1);
- Nizova - uređena lista vrijednosti; u programskim jezicima se najčešće realizuje kao niz, vektor ili lista. Zapis počinje sa otvorenom ugaonom zagradom, nakon čega ide niz vrijednosti razdvojen zarezima, a završava se zatvorenom ugaonom zagradom.

Vrijednosti u oba slučaja mogu biti: objekat, niz, niska (engl. *string*), broj, nepostojeće (engl. *null*) vrijednosti, istinite i neistinite (engl. *true* i *false*) vrijednosti. Broj i niska se zapisuje na isti način kao u C ili Java jeziku.

```
{
      id: 5,
      brTel: "38269295646",
      ime: "Petar Rutesic",
      pass: "25d55ad283aa400af464c76d713c07ad",
      pol: 1,
      email: null,
      datumReg: "1467497298",
```

```
    status: 1,
    token: "38dae70e06f7c25d7a1829d42358c8d9a91cb457",
    resetToken: "oXNymfoL6B",
    deviceToken: "cdaTon4KldA:APA91bEApicwJZhDAv38hAFEY3_HMz3IJpGi
y0m2YkaQE4fhoBEljkhcgY9ualx831v2jfoOlk56eu4ZU9YPBMJfzBSXV5RewMslIB84
gtWbw4ci2TF42uh4_-5TAdriPKB4nKGx5fc",
    uspjesno: 1
}
```

Primjer 1 – *JSON* niska

2.4.3 XML

Proširivi jezik za označavanje, *XML*(engl. *eXtensible Markup Language*), propisan je od strane W3C i predstavlja format donekle sličan *HTML*-u. *XML* se sastoji od elemenata ukalupljenih u formatu <oznaka>element</oznaka>, gdje svaki element može imati određene atribute. Na primjeru 2 dat je izgled *XML* datoteke sa podacima o jednoj mački. Očigledno je da je ovaj skup podataka lako proširiv. Oznake mogu sadržati i mala i velika slova.

```
<?XML version="1.0"?>
<CAT>
    <NAME short="Iz">Izzy</NAME>
    <BREED>Siamese</BREED>
    <AGE>6</AGE>
    <ALTERED>yes</ALTERED>
    <DECLAWED>no</DECLAWED>
    <LICENSE>Izz138bod</LICENSE>
    <OWNER>Colin Wilcox</OWNER>
</CAT>
```

Primjer 2 - *XML* datoteka

Podaci u *XML* datoteci se mogu vizualizovati kao drvo. U primjeru 2, korijeni element bi bio *CAT,* dok su listovi *NAME, BREED* itd. Ovo je značajno prilikom parsiranja *XML* datoteke parserima zasnovani na objektom modelu dokumenta, DOM-u (engl. *Document Object Model*). Postoje dvije vrste parsera [2]:

- DOM zasnovani parseri – tumače *XML* kao ugniježdenu strukturu. Prednost ovakvih parsera je što nakon parsiranja omogućavaju nasumični pristup podacima. Nedostatak parsera je kod učitavanja velikog ali jednostavnog *XML*-a, jer je često potrebno čitavu datoteku učitati u memoriju što može biti prilično skupo.
- SAX ili povlačeći (engl. *pull*) parseri – sve strukture u *XML* fajlu se smatraju događajima, a sam *XML* fajl tokom (engl. *stream)*. Parser ide redom od početka dokumenta i učitava jedan po jedan događaj. Ovo omogućava manju potrošnju memorije jer je u jednom trenutku aktivan najviše jedan događaj, a ukoliko je baš potrebno, moguće je ručno formirati neku strukturu koja bi bila optimalnija od one koja se dobija DOM parserima

XML je derivat *SGML*-a (engl. *Standard Generalized Markup Language*), koji je nastao kao potreba da se na lak način vrši označavanje podataka u tehničkoj dokumentaciji. Mane *SGML*-a su bile preopširnost i složenost, što je dalje uzrokovalo da nije bio široko raširen. Jedan od derivata *SGML*-a je i *HTML*. Osnovna razlika je što oznake kod *XML*-a definišu strukturu podatka dok kod *HTML*-a definišu izgled podataka – gdje se i kako nalaze koji podaci [8].

XML danas služi i za transport i skladištenje podataka. Prva verzija *XML* 1.0 je standardizovana 1998. godine i zasnovan je na *Unicode* 2.0. Kako se *Unicode* razvijao i podržavao sve više jezika i pisama, 2004. godine je objavljen *XML* 1.1.

Prednosti *XML*-a su što je lako razumljiv računaru, međutim zbog redudantnih oznaka i opširne sintakse njegova obrada može biti računski skupa, naročito u poređenju sa *JSON*-om. *XML* je razumljiv i čovjeku, ali često ga komplikovana sintaksa može zbunjivati. Većina programskih jezika današnjice ima parsere za *XML* datoteke.

2.4.4 URI

Uniformni identifikator resursa, *URI* (engl. *Uniform Resource Identifier*) je niska karaktera koja služi da identifikuje resurs. Najpoznatiji oblik *URI*-a jeste *URL* (engl. *Uniform Resource Locator*) koji se naziva još i veb adresa. *URL* predstavlja oblik *URI*-a iako je kao koncept nastao u početnoj fazi razvoja veba, prije *URI*-a. Format *URI*-a je dat sledećim šablonom [9]:

```
šema:[//[korisnik[:lozinka]@]host[:port]][/putanja][?upit][#fragment]
```

- Neke od najpoznatijih šema su *HTTP*(s), ftp, irc, mailto, file, data;
- Dio za autentifikaciju korisnika je dat opcionim elementima korisnik i lozinka;
- Host označava registrovano ime ili IPv4 ili IPv6 adresu;
- Port označava broj porta na kom je dostupan host;
- Putanja je data kao putanja do resursa, u kojoj postoji hijerarhija u kojoj se nivoi definišu sa kosom crtom – nalik kod sistema za upravljanje datotekama;
- Upit – služi za upit nad nehijerarhijskim podacima, najčešće skup parova atribut=vrijednost razdvojenih sa & ili “;”;
- Fragment – označava dio resursa, kod *HTML*-a je to najčešće identifikator (ID) *HTML* elementa.

2.5 Arhitektura orijentisana na resurse

REST je donekle nastao kao pokušaj prevazilaženja komplikovanosti *SOAP* servisa. Za razliku od *SOAP*-a gdje je potrebno korišćenje posebnih omotača, opisivača servisa i sličnih stvari, *REST*-servisi se oslanjaju samo na *HTTP*. Da bi servis bio *REST*-oliki, uvodi se pojam arhitekture orijentisane na resure, ROA (engl. *Resource*

oriented architecture). U svojoj doktorskoj disertaciji [6], Roj Filding je naveo četiri osnovna koncepta arhitekture orijentisane na resurse:

1. Resursi;
2. Njihova imena (*URI*) ;
3. Njihove reprezentacije;
4. Veze (engl. *links*) među njima.

i četiri osobine:

1. Adresabilnost;
2. Nepostojanje stanja;
3. Povezanost;
4. Uniformni interfejs.

2.5.1 Resursi

Glavna apstrakcija informacije kod *REST*-a je resurs. Resurs označava nešto što je dovoljno bitno da bi moglo biti označeno kao resurs. Resurs je najčešće nešto što može biti sačuvano u računaru i reprezentovano putnem niza bajtova: dokumenti, tabela u bazi podataka, rezultat rada algoritma itd [2].

Resursima se mogu smatrati i fizički objekti kao npr. jabuka ili apstraktni koncepti kao poštenje, ali je reprezentacija ovakvih resursa problematična.

2.5.2 Imena resursa

Resurs mora imati bar jedno ime, tj. uniformni identifikator resursa - *URI*. Svaki *URI* označava tačno jedan resurs, a isti resurs može imati više *URI*-a. Neki od primjera *URI*-a su:

1. HTTP://192.168.0.1/admin/registar/dozvole/
2. HTTP://www.uriprimjer.com/admin/dashboard/
3. HTTP://www.uriprimjer.com/blog/2010/12/5/0
4. HTTP://www.uriprimjer.com/program/verzija1_5/
5. HTTP://www.uriprimjer.com/program/zadnja_verzija/

URI-i pod brojem 4. i 5. mogu označavati isti resurs (kada je verzija 1.5 ujedno i zadnja verzija programa). Uniformni identifikatori resursa tehnički ne moraju da imaju neku strukturu ili predvidivost, ali je poželjno.

URI bi trebalo da označava ime resursa a ne akciju koja se vrši nad njim, jer je akcija definisana metodom (najčešće *HTTP* metodom) ili parametrima.

2.5.3 Reprezentacija resursa

Resurs je izvor reprezentacije, a reprezentacija je samo neka informacija o trenutnom stanju resursa. Postoje različite vrste reprezentacije resursa – *HTML* strana, *XML* dokument, *JSON* dokument, *TXT* dokument itd. Iako su reprezentacije različite, radi se o istom resursu.

Najčešće server šalje reprezentacije klijentu, međutim, u slučaju da klijent želi da vrši izmjenu postojećeg ili pravljenje novog resursa na serveru, onda on mora poslati serveru odgovarajuću novu reprezentaciju.

U poglavlju 2.4.1 je definisan pojam pregovaranja o sadržaju kod *HTTP* odgovora. U slučaju reprezentacije resursa, klijent može da pošalje serveru koje sve reprezentacije resursa može da razumije, a server može da mu vrati neku od odabranih reprezentacija. Dobra stvar kod *REST*-a u odnosu na *SOAP* je ta što reprezentacija nije fiksirana, već klijent može sam da bira koji vid reprezentacije želi (*JSON*, *XML*), dok je kod *SOAP*-a to bio najčešće *XML*, bez mogućnosti izbora i sa precizno definisanim formatom.

2.5.4 Adresabilnost

Aplikacija je adresabilna ako otkriva interesantne aspekte svojih podataka kao resurse koji se mogu čitati, mijenjati, praviti i brisati. Pošto su resursi dostupni preko *URI*-a, adresabilna aplikacija obezbjeđuje *URI* za svako parče informacije koje može da prikaže na razumljiv način [2].

Iz perspektive krajnjeg korisnika, adresiranje je najvažniji aspekt svakog web sajta ili aplikacije. Npr. *URI* za rezultate google pretrage za ključnu riječ “vw buba” glasi: https://www.google.com/search?q=vw+buba. Ukoliko *HTTP* ne bi bio adresabilan, drugi korisnik ne bi mogao vidjeti rezultate pretrage sve dok u pregledač veba ne unese adresu Gugl sajta, pa zatim unese ključnu riječ i klikne dugme za pretragu. Adresabilnost omogućava da kopiranjem *URI*-ja u veb pregledač dobijemo isti rezultat kao da smo već obavili niz koraka koji su nas prvobitno doveli do tog *URI*-a.

Moguće je ulančavati *URI*-je (engl. *chaining URIs*) i koristiti jedan *URI* kao ulazni parametar za drugi. Na primer, možemo koristiti neki eksterni veb servis kako bismo validirali *HTML*, ili da bismo preveli tekst na drugi jezik. Ovi veb servisi očekuju *URI* kao ulazni parametar. Da *HTTP* nije adresabilan, ne bismo imali načina da im kažemo koje resurse želimo da obradimo.

Zahvaljujući adresiranju, klijenti mogu lako da koriste veb sajtove. Korišćenje adresabilnosti donosi aplikaciji i njenim korisnicima mnoge pogodnosti *REST*-a.

2.5.5 Nepostojanje stanja

Jedna od važnih osobina ROA-e je nepostojanje stanja. Nepostojanje stanja je jedna od ključnih osobina i *HTTP* protokola. Stanje se u odnosi na stanje aplikacije, koje je različito od stanja resursa (stanja resursa su ista za sve klijente i ona vezana za server, dok stanja aplikacije ne moraju biti ista i nalaze se na klijentu). Šta ovo znači u praksi?

Na primjer, neka klijent šalje zahtijev serveru, koji se sastoji od toga da želi da vidi podatke za prvih 30 radnika. Server opslužuje zahtijev i vraća rezultat klijentu. Ukoliko bi klijent želio da vidi podatke za sledećih 30 radnika, klijent mu mora opet poslati zahtijev. Postoje dva moguća scenarija:

- Ukoliko nema stanja, klijent mora da šalje sve potrebne parametre da bi server mogao da obradi zahtijev [6]. U konkretnom slučaju klijent bi morao poslati indeks od kojeg počinju da se prikupljaju podaci za 30 radnika (0,30,60 itd) – ovakav je slučaj sa *HTTP*-om;
- Ukoliko ima stanja, server bi znao ko su prethodnih 30 radnika čije je podatke već isporučio i automatski bi mu proslijedio sledećih 30 .

Korišćenje kolačića (engl. *cookies*) preko kojih bi se kontrolisalo stanje odstupa od načela nepostojanja stanja. Korišćenje kolačića u ovom slučaju dolazi u obzir jedino ako se njima održava sesija i vrši autorizacija. Dakle, klijent dobija akreditive od nekog servisa, i potrebno je da svaki put zajedno sa zahtjevom šalje te akreditive, jer ne postoje stanja. Ovdje se javlja problem transporta akreditiva, jer slanje šifre i korisničkog imena putem *HTTP*-a je prilično osjetljivo na špijuniranje.

Jedan od primjera protokola sa stanjem bi bio protokol za upravljanje datotekama. U svakom trenutku se zna stanje, tj. trenutni direktorijum. Da bi se učitao fajl iz trenutnog direktorijuma, nije potrebno opet kucati komandu kojom se dolazi do trenutnog direktorijuma – informacija o trenutnom direktorijumu već postoji i ona predstavlja stanje.

Uvođenje stanja u *HTTP* protkol bi uveliko povećalo njegovu kompleksnost iako bi zahtjevi bili jednostavniji.

Osobina nepostojanja stanja dovodi do poboljšanja:

- Vidljivosti – server ne mora da razmatra ništa osim trenutnog zahtijeva;
- Skalabilnosti – ne moraju se dijeliti stanja između zahtijeva, pa se oni brže obrađuju;
- Pouzdanost – dolazi do lakšeg oporavljanja prilikom djelimičnih podbačaja.

Ipak, ova osobina ima i neke negativne strane, a to je da se često iste informacije moraju slati svakog puta iako ih zahtijevi dijele. Druga negativna pojava je da server ima smanjenu kontrolu nad konzistentnim ponašanjem aplikacije, u smislu da aplikcija zavisi od korektne implementacije semantike u klijentima.

2.5.6 Povezanost

Pojam povezanosti u kontekstu ROA arhitekture označava mijenjanje stanja prateći veze unutar hipermedije. Hipermedija je ne samo serijalizovana struktura podataka, već i dokument koji sadrži veze ka drugim resursima.

Trenutno stanje *HTTP* sesije se ne čuva na serveru, već se prati na klijentu kao stanje aplikacije, a napravljeno je putanjama koje klijent prati na veb-u.

Kao tipičan primjer povezanosti se može uzeti Guglov pretraživač veba. Nakon što korisnik unese ključne riječi, dobija hipertekst dokument koji sadrži podatke i veze ka drugim resursima. Unutar tog dokumenta osim podatka ima veze ka sledećem stanju (engl. *next results*), link ka Guglovoj keširanoj verziji resursa i sl. Sve ove veze predstavljaju susjedna stanja trenutnom stanju. Dakle, klijent prelazi u različita stanja praćenjem hiper-veza (engl. *hyperlink*) ili popunjavanjem *HTML* formi.

2.5.7 Uniformni interfejs

U principu, postoje četiri osnovne operacije koje se mogu vršiti nad nekim resursom, definisane kao CRUD (u poglavlju 2.2).

Za *CRUD* operacije se koriste četiri *HTTP* metoda:

- Pravljenje novog resursa – metod *POST*;
- Dobijanje reprezentacije resursa – metod *GET*;
- Izmijena postojećeg resursa – metod *PUT* (ukoliko resurs ne postoji ponaša se kao *POST*) ;
- Brisanje postojećeg resursa – metod *DELETE*.

Manje korišćene metode su : *HEAD* (vraća zaglavlje koje se dobija kroz *GET*) i *OPTIONS* (vraća spisak raspoloživih metoda nad resursom).

Ukoliko se prate upustva za uniformni interfejs, dobijaju se dvije dobre osobine:

- Idempotentnost i
- Bezbijednost.

Idempotentnost je princip zastupljen u matematici. Idempotentnost znači da ukoliko izvršimo neku operaciju jednom, svako naredno izvršavanje iste operacije ne mijenja rezultat. Npr. idempotentna operacija je množenje sa nulom: 4*0 je isto što i 4*0*0*0. Množenje sa jedinicom je i idempotentno i bezbijedno 4*1 = 4 = 4*1*1*1.

Bezbijednost znači da izvršavanje operacije ne mijenja rezultat – npr. operacija čitanja promjenljive koja se nalazi na nekoj memorijskoj lokaciji.

Metodi *PUT* i *DELETE* su idempotentni. Ukoliko se napravi novi resurs korišćenjem *PUT* metode, i nakon toga pošalje identičan *PUT* zahtjev, resurs ostaje isti kao i nakon pravljenja. Ukoliko se promijeni stanje *PUT* zahtjevom, slanje identičnog

zahtjeva ne mijenja stanje resursa ponovo. Prvim brisanjem metodom *DELETE*, resurs nestaje. Ukoliko se opet izvrši brisanje, resursa i dalje nema

Metodi *GET* i *HEAD* su bezbijedni i idempotentni. Čitanje podataka zahtjevom *GET* ne mijenja resurs. Nema nikakve veze da li je klijent napravio jedan, deset ili nijedan *GET* ili *HEAD* zahtjev. Jasno je da klijent slanjem ovakvih zahtjeva ne može da napravi nikakvu štetu, iako postoje određene nuspojave (npr. ako postoji brojač pregleda koji se uvećava za jedan sa svakim *GET* zahtjevom).

Metod *POST* metod nije ni idempotentan ni bezbijedan. U slučaju slanja jednog *POST* zahtijeva, slanje narednog bi promijenilo stanje (moglo bi kreirati novi resurs).

3 Veb soketi

Potreba za modernim veb aplikacijama je uzrokovala razvoj novih i usavršavanje postojećih tehnologija. U vrijeme nastanka, veb je zamišljen kao jednostavan sistem u kome je komunikacija zasnovana na principu jedan zahtijev klijenta - jedan odgovor servera. Kao rezultat potebe za ovakvim načinom komunikacije, rodio se *HTTP*. Međutim, mnogo toga se promijenilo od nastanka veba pa do danas.

Razvojem veb aplikacija, javila se potreba za istovremenom dvosmijernom komunikacijom između klijenta i servera. Možda najbolji primjer aplikacija koje koriste istovremenu dvosmijernu komunikaciju jesu aplikacije za razmijenu poruka (engl. *chat*) ili video igre za više igrača (engl. *multiplayer games*). Jasno je da *HTTP*/1.1 sam po sebi ne može biti optimalno riješenje za ovakav scenario.

3.1 Tehnologije potiskivanja i povlačenja

Tehnologija potiskivanja predstavlja koncept suprotan tehnologiji povlačenja. Kod tehnologije povlačenja prenos podataka od servera ka klijentu se vrši isključivo nakon što klijent pošalje zahtjev serveru. U slučaju tehnologije potiskivanja, komunikacija od servera ka klijentu ne mora da počinje klijentskim zahtjevom. Već pominjani *HTTP*/1.1 najbolje opisuje povlačenje – svi podaci od servera ka klijentu dolaze nakon slanja zahtjeva serveru od strane klijenta.

Potrebno je napomenuti da *HTTP*/2.0 donosi mogućnost isporučivanja podataka od strane servera [5], bez prvobitnog zahtijeva klijenta. Kod ranijih verzija *HTTP* protokola pregledač veba preuzima *HTML* dokument i u njemu traži koji su još resursi potrebni da se preuzmu od servera – slike, skriptovi i slično. Nakon toga on za svaki od resursa mora da pošalje zahtijev ka serveru, i to najviše jedan zahtijev istovremeno. U novoj verziji protokola, server isporučuje hipertekst i zajedno sa njim i potrebne slike, skriptove, objekte i sve što je neophodno za prikaz na klijentovom pregledaču veba. Time se smanjuje vrijeme učitavanja veb strane. Ipak, u trenutku pisanja ovog rada, *HTTP*/2 još uvijek nije široko prihvaćen.

Postoje načini da se kroz *HTTP*/1.1 simulira serversko potiskivanje podataka. Neki od njih su:

- *XHR Polling* – konstantno slanje zathijeva u određenim vremenskim intervalima ka serveru od strane klijenta. Očigledno je da je ovakav pristup traćenje resursa, jer u velikom broju slučajeva server neće imati da pošalje nikakvu novu informaciju. Najbolji primjer za to bi bio upravo kod aplikacija za razmjenu instant poruka – moguće je da korisnici međusobno razmijenjuju poruke dosta rijetko, na svakih 5 minuta, a da se *polling* vrši na svakih 10 sekundi. U tom slučaju oko 28 do 29 zahtijeva za povlačenje nikakvu novu informaciju nisu donijeli. Takođe, ovakav pristup je loš za aplikacije kod kojih je bitno da je komunikacija u realnom vremenu – u navedenom primjeru

moguće je da se javlja čekanje i do blizu 10 sekundi da se dobije neki podatak. Primjer je dat na slici 4a;

- *Long polling* – varijanta *polling*-a u kojoj server ne šalje odgovor ako nema ništa novo da pošalje, ili pošalje samo zaglavlje *HTTP* odgovora i ostavlja klijenta da čeka nastavak. Kada server želi da proslijedi novu informaciju klijentu, on samo nastavlja da odgovara koristeći postojeću otvorenu konekciju. Nakon primanja odgovora, klijent šalje novi *HTTP* zahtjev i čeka. U ovom slučaju se eliminiše čekanje koje se javlja kod tradicionalnog *pooling*-a, tj. vrijeme koje protekne od trenutka kad server raspolaže sa novom informacijom do vremena kad klijent pošalje *polling* zahtijev. Primjer je dat na slici 4b;
- *HTTP streaming* – različite tehnike koje omogućavaju serveru da šalje više od jednog odgovora jednom klijentu, najčešće u djelovima (engl. *multipart/chunkedi*). W3C ga je standardizovao kao događaje koje šalje server, *SSE* (engl. *Server-Sent Events*) korišćenjem *text/event-stream MIME* tipa. Još jedan od primjera je *MIME* tip sadržaja *multipart/x-mixed-replace* koji je svojevremeno uveo *Netscape* pregledač veba – taj *MIME* tip označava da server može poslati novu verziju slike veb pregledaču koji nakon toga mijenja postojeću sliku, čime bi se mogao simulirati prenos videa. *API* veb pregledača za *SSE* se zove *EventSource API*. Primjer je dat na slici 4c;
- *Comet/server push* – generalni izraz za tehnike koje koriste *Long polling* i *HTTP streaming*, trudeći se da podrže što više veb pregledača i veb servera;
- *WebSocket* – transportni sloj zasnovan na *TCP*-u koji koristi *HTTP Upgrade handshake* [10]. Za razliku od *TCP*-a koji je transport sa tokom, *WebSocket* je transport zasnovan na porukama, u smislu da se poruke moraju u potpunosti kompletirati prije nego ih aplikacija primalac ima na raspolaganju. Razmjena podataka između klijenta i servera se nakon inicijalnog *HTTP* para zahtjev - odgovor može izvršavati asinhrono. Standardizovan je u standardu *HTML5*, a *API*-je imaju gotovo svi moderni veb pregledači. Više riječi o *WebSocket*-ima će biti u nastavku rada. Primjer je dat na slici 4d.

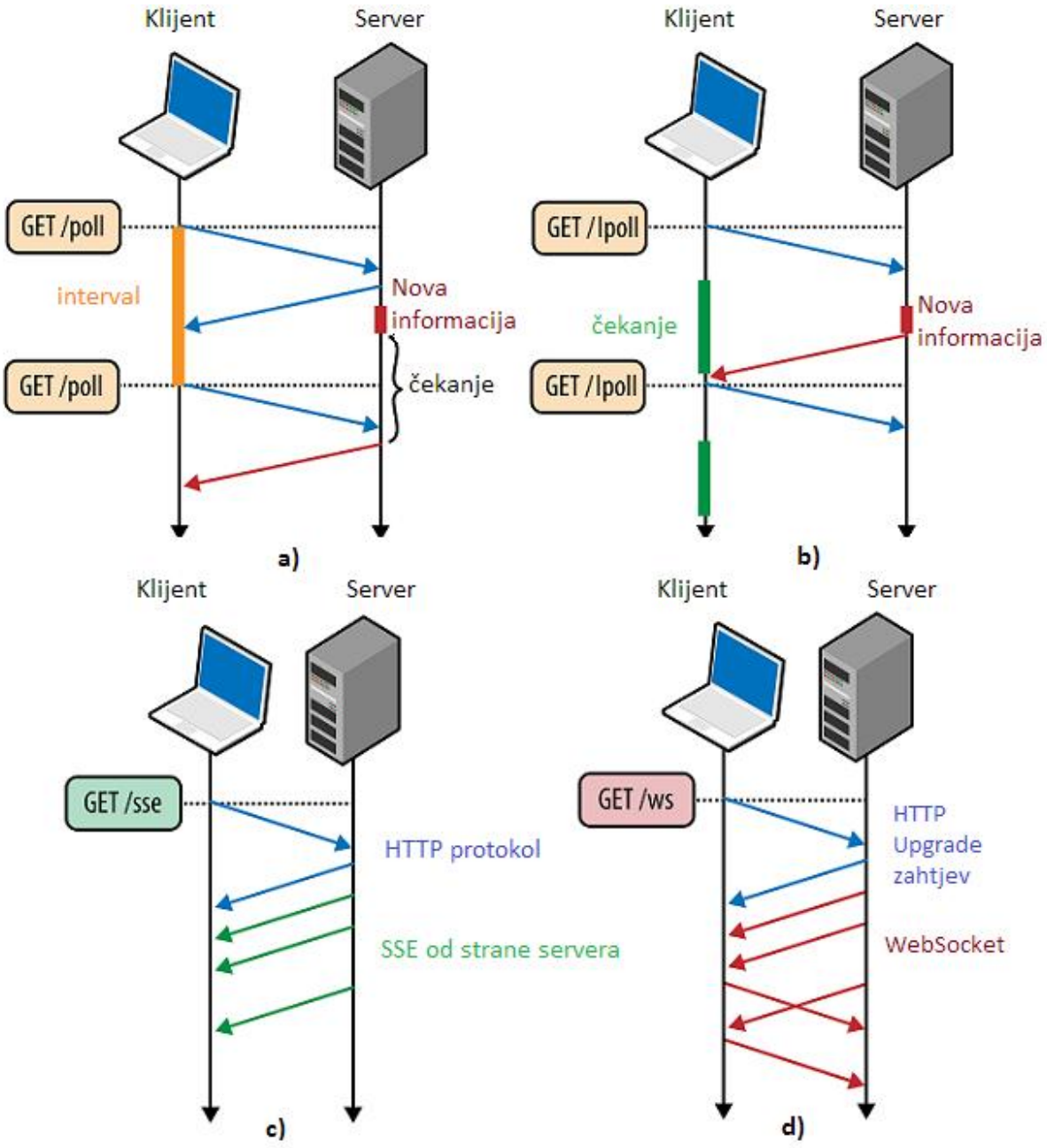
U vrijeme pisanja ovog rada, IETF radi na nacrtu standarda za protokol *WebPush* koji koristi *HTTP*/2, a koji bi u realnom vremenu mogao da isporučuje događaje klijentu, kao što su dolazeći pozivi, poruke i slično [11].

3.2 Soketi

Da bi dva procesa na različitim mašinama razmijenjivala podatke, potrebno je da se nekako međusobno povežu. Način za to je mrežni soket (engl. *network socket*).

U oblasti interneta mrežni soket predstavlja internu krajnju tačku u nekom čvoru, koja služi za slanje i primanje podataka. Soket bi se na našem jeziku mogao prevesti kao utičnica, međutim, radi jednostavnosti, u radu će se koristiti termin soket.

Termin soket se koristi zbog sledeće analogije: soketi predstavljaju utičnice za struju (krajnje tačke u čvorovima), a komunikacija između njih se odvija dok su povezane kablom na čijim krajevima se nalaze utikači.



Slika 4 - Primjeri za *polling*, *long polling*, *SSE* i *WebSocket*

Razmjena podataka se u slučaju soketa vrši na sledeći način: prave se soketi na mašinama koje razmijenjuju podatke. Pri pravljenju soketa definiše se opisivač soketa (engl. *socket descriptor*), koji služi da precizno identifikuje soket na mašini. U praksi je ovo obično neki cijeli broj. Soketu može da pristupi samo ona mašina na kojoj se nalazi soket, u smislu da opisivač soketa ne može da koristi druga mašina.

U slučaju internet protokola *IP*, i protokola *TCP* ili *UDP*, koristi se termin internet soket. Proces A pravi soket za neku *IP* adresu i port, npr. 1.2.3.4:8888, i dobija

opisivač soketa, npr. 132. Neka taj proces želi da komunicira sa procesom B na adresi 4.3.2.1:1111 koja ima opisivač 200. Sav saobraćaj koji dođe sa adrese 4.3.2.1:1111 ka procesu A se preusmjerava na soket 132, a proces A sve podatke preusmjerava na soket 132 kada želi da ih šalje ka procesu B. Oni međusobno ne mogu da vide opisivače soketa. Pravljenje soketa na *IP* adresi i portu se još naziva i vezivanje (engl. *binding*). Internet soketi su *API*-ji koji su podržani od strane operativnog sistema. U zavisnosti od transportnog protokola, razlikuju se tri vrste soketa:

- *Datagram* soketi – koriste *UDP* protokol, kod koga nije bitna pouzdanost isporuke svih paketa (kao ni redosled)
- *Stream* soketi – najčešće koriste *TCP* protokol
- Sirovi soketi – koriste samo *IP* protokol, proces čak vidi i zaglavlje *IP* paketa jer nema transportnog sloja. Takođe, nemaju port kao kada se koriste transportni protokoli *UDP* ili *TCP*.

U slučaju klijent – server arhitekture, dešava se specifičan slučaj da je više klijenata povezano na istu adresu i port servera. Kod s*tream* soketa, problem se rešava tako što server pravi djecu procese za koje pravi njihove sopstvene sokete, koji komuniciraju sa klijentima. Dakle, svaki od djece procesa preko svog soketa komunicira sa svojim klijentom, a svi ti soketi dijele istu adresu i port na serveru. Nakon što napravi soket, server osluškuje saobraćaj na soketu. Kada se završi razmijena podataka, zatvara se soket. Primjer komunikacije soketom između klijenta i servera je dat na slici 5.

3.3 WebSocket

WebSocket-i omogućavaju istovremenu dvosmijernu (engl. *full-duplex*) razmjenu poruka između klijenta i servera. Poruke mogu biti tekstualne ili binarne. *WebSocket*-i su više od mrežnih soketa, sa obzirom da veliku većinu stvari pregledač veba apstrahuje kroz veoma jednostavan *API* koji omogućava dodatne usluge [5]:

- Pregovaranje o konekciji i primjena politike istog porijekla (engl. *same origin policy*);
- Interoperabilnost sa postojećom *HTTP* infrastrukturom;
- Komunikaciju orijentisanu na poruke sa efikasnim uramljivanjem poruka;
- Pregovaranje o potprotokolu.

Kod *WebSocket*-a se jasno razlikuju klijent i server. Na serveru je pokrenut *WebSocket* server. On može da komunicira samo sa klijentom, tj. da odgovara na njegove zahtijeve. Takođe, zahvaljujući dvosmijernoj vezi moguće je podesiti server tako da služi kao centralni posrednik u komunikaciji između dva klijenta.

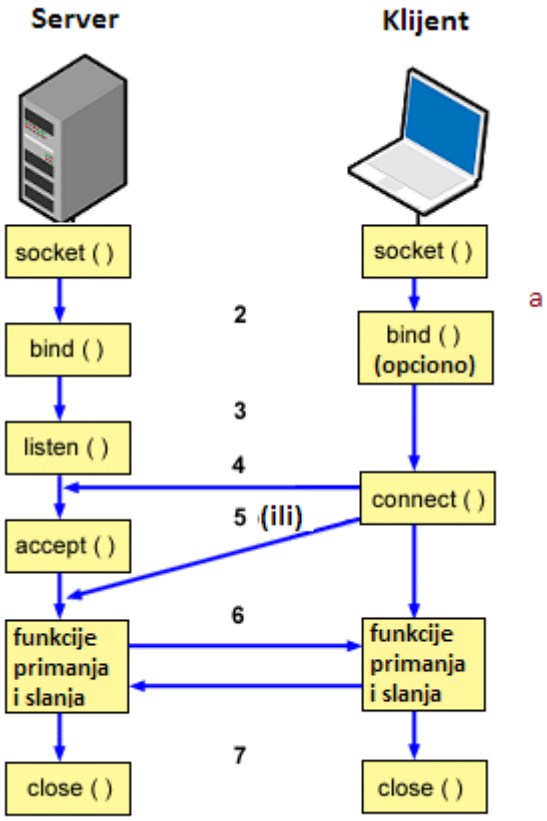
Poruke se efikasno razmijenjuju, redudantnost podatka pri prenosu je minimalna čime se u gotovo realnom vremenu vrši razmijena podataka. Važno je napomenuti da *WebSocket*-i nisu zamijena za *XHR* ili *SSE*, kao ni za *HTTP*.

WebSocket-i su skup više standarda: *WebSocket API* je standardizovao W3C, dok je sam protokol (RFC 6455) definisan od strane HyBI Radne grupe (IETF).

WebSocket protokol koristi drugačiju šemu u *URI*-u od *HTTP*-a:

- U slučaju običnog *WebSocket* -a, *URI* je ***ws://adresa:port/soket***;
- U slučaju sigurnog *WebSocket* -a koji koristi TSL sigurnosni sloj, šema je ***wss*** umjesto ***ws***.

Kako je *WebSocket API* dio *HTML*5 standarda, većina modernih pregledača veba ga podržava.



Slika 5 - Soketi i klijent - server arhitektura

Neki od pregledača veb-a za personalne računare koji podržavaju *WebSocket*-e su [10]:

- Internet Explorer 10+;
- Mozilla Firefox 11+;
- Google Chrome 16+;
- Safari 6+;
- Opera 12+.

Takođe, podrška je razvijena i za veb pregledače za mobilne uređaje kao i za nativne (engl. *native*) aplikacije, kroz različite *API*-je za odgovarajuće platforme.

3.3.1 Protokol

Sam *WebSocket* protokol se sastoji od dvije glavne komponente: jedna je početno *HTTP* rukovanje (engl. *handshake*) kojim se uspostavlja konekcija a druga je binarno uokvirivanje (engl. *binary framing*) poruka da bi se omogućila mala redudantnost prilikom razmjene teksta ili binarnih podataka [5].

Protokol je rezultat želje da se riješe problemi dvosmijerne komunikacije u postojećoj *HTTP* infrastrukturi, pa zato koristi portove 80 i 443 koje koristi i *HTTP*. Iako je to njegova prvobitna namjena, protokol se može koristiti nezavisno od pregledača veba, dakle, ne mora se koristiti samo za dvosmijernu komunikaciju kod veb aplikacija [12].

Razmjena poruka se vrši tako što pošiljalac dijeli tekstualnu ili binarnu poruku u nekoliko okvira (engl. *frames*), a primalac vidi poruku tek nakon što se svi okviri sjedine. Da bi to bilo izvodljivo, koristi se sloj za binarnog uramljivanja poruka. Ova podjela i sastavljanje su nevidljivi za aplikaciju. Zahvaljujući tome što pri razmijeni nema zaglavlja, već se sve nalazi u u okvirima, redudantnost pri razmjeni poruka je izuzetno mala i iznosi od 2 do 14 bajtova po okviru.

Postoje određena proširenja za *WebSocket* protokol, kao što su proširenje za multipleksiranje koje omogućava da više *WebSocket* konekcija dijeli istu transportnu konekciju, ili proširenje za kompresiju koje omogućava kompresiju poruka.

3.3.2 Klijent

Klijentska aplikacija koristi veoma jednostavan *WebSocket API* koji daje pregledač veba ili biblioteka u programskom jeziku. *API* brine o upravljanju konekcijom i razmijeni poruka, pa je korišćenje *WebSocket*-a još jednostavnije od korišćenja standarnih mrežnih soketa.

Konekcija započinje pravljenjem novog *WebSocket* objekta i opcionim odabirom potprotokola. Neki od bitnih atributa *WebSocket* objekta su:

- ***binaryType***: niska koja govori da li se radi prenos binarnih podataka;
- ***readyState***: konstanta koji govori o stanju konekcije, može biti *CONNECTING*:0, *OPEN*:1, *CLOSING*:2, *CLOSED*:3;
- ***onopen***: JavaScript funkcija koja se poziva nakon uspostavljanja konekcije, tj. nakon što *readyState* postane *OPEN*;
- ***onmessage***: JavaScript funkcija koja se poziva nakon što klijent primi poruku
- ***onerror***: JavaScript funkcija koja se poziva kada dođe do greške. Nakon nje obično slijedi *onclose;*
- ***onclose***: JavaScript funkcija koja se poziva kada se zatvori konekcija, bez obzira da li je prekid normalan ili usled greške. Iz JavaScript *Event* objekta se može saznati usled čega je došlo do prekida konekcije.

Metode *WebSocket* objekta su:

- ***WebSocket.send(Tip Poruka)*** – šalje poruku ka serveru. Tip može biti *Blob*, *ArrayBuffer* ili *DOMString;*
- ***WebSocket.close(unsigned short kod, DOMString razlog)***: zatvara konekciju, argumenti su opcioni i označavaju razlog zatvaranja konekcije.

Potprotokol koji se opciono bira pri uspostavljanju konekcije služi da bi klijent i server međusobno mogli da tumače poruke. Sama poruka može biti jednog od tri tipa – *Blob*, *ArrayBuffer* ili *DOMString*, ali šta ako se recimo unutar *DOMString* niske nalazi *JSON* objekat? *WebSocket* poruke nemaju metapodatke kao što imaju *HTTP* poruke, pa se ovaj problem može riješiti korišćenjem potprotokola.

3.3.3 Server

U najvećem broju slučajeva serverska implementacija *WebSocket*-a ima iste metode kao i klijentski *API*. Sam server može biti pokrenut u okviru veb servera ili kao zasebna aplikacija.

Postoji veliki broj riješenja koja se mogu koristiti, implementiranih u različitim jezicima ili razvojnim okvirima. Među najpopularnjima su *SocketIO* napisan u JavaSkriptu, *Tornado* ili *Autobahn* napisani u Pajtonu, *Ratchet* napisan u *PHP*-u itd.

Prilikom pokretanja servera, isti se veže za adresu i port. Nakon toga on čeka na zahtjev klijenta za uspostavljanje konekcije. Spisak svih klijenata koji su trenutno povezani sa serverom se čuva najčešće u obliku neke kolekcije, niza ili tome slične strukture. On se ažurira svaki put kada novi klijent pošalje zahtijev za uspostavljanje konekcije i kada se raskida konekcija. Svaka od implementacija sadrži bar metode koje se pozivaju u prilikom:

- uspostavljanja konekcije sa klijentom;
- raskida konekcije između klijenta i servera;
- prilikom prijema klijentskih poruka.

Server, kao i klijent, može svojevoljno da raskine konekciju sa klijentom ili da mu šalje poruke. U zavisnosti od implementacije, postoje i dodatne metode, kao što su na primjer slanje ping pong poruka čime se provjerava aktivnost konekcija i slično. Server može da se ponaša i kao posrednik između dva klijenta koja komuniciraju, čime krajnji korisnici mogu imati utisak da se radi o direktnoj klijent - klijent komunikaciji. U ovom slučaju on služi uglavnom za usmjeravanje poruka.

3.3.4 Tipovi poruka

Poruke koje se razmijenjuju preko *WebSocket*-a mogu biti u različitim formatima. Sam *WebSocket* protokol podržava tri vrste poruka:

1. *BLOB*, (engl. *Binary Large OBject*)– ovaj format podataka je efikasan ako je potrebno prenijeti sirove binarne podatke koji će biti samo reprezentovani na klijentu, npr. prenos slike koja se nakon toga učita u <img> tag
2. *ArrayBuffer* – ova binarna struktura je pogodna za transport strukturiranih binarnih podataka. Takvi podaci imaju poredak i moguće je pristupiti određenim djelovima podataka. Npr. moguće je da se razmijenjuju poruke koje sadrže dvodimenzionalnu strukturu gdje je prvi element identifikator pošiljaoca koji primalac vidi, a drugi element sama poruka. U tom slučaju se razmijenjuje binarna reprezentacija ove strukture, a onda se ista tumači nakon što stigne do primaoca
3. Niska (engl. string) – prenos tekstualnih poruka. Ove poruke mogu imati i određenu strukturu, npr. *JSON* ili *XML*. Takve poruke se po pristizanju parsiraju. Za definisanje te strukture mogu se koristiti potprotokoli, npr. kao što je *JSON-RPC*

3.3.5 Bezbijednost

Za *WebSocket* je izuzetno bitno odakle dolazi zahtjev za uspostavljanje konekcije. Kao rezultat toga je uveden parametar zaglavlja *Origin*. Postoji još nekoliko parametara zaglavlja koja dozvoljavaju klijentu da se nadogradi na protokol *WebSocket*, a ona počinju sa prefiksom *Sec-* i garantuju da će svaki *WebSocket* zahtijev biti inicijalizovan preko *WebSocket* konstruktora, a ne preko *HTTP API*-ja koji možda žele da pristupe porukama koje se razmijenjuju [10].

HTTP Upgrade zahtijev koji se šalje prilikom uspostavljanja konekcije je dat na primjeru 3, a odgovora na primjeru 4.

```
GET /socket HTTP/1.1
Host: thirdparty.com
Origin: HTTP://example.com
Connection: Upgrade
Upgrade: websocket
Sec-WebSocket-Version: 13
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Protocol: appProtocol, appProtocol-v2
Sec-WebSocket-Extensions: x-webkit-deflate-message, x-custom-extension
```

Primjer 3 - Uspostavljanje Websocket konekcije

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Access-Control-Allow-Origin: HTTP://example.com
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: appProtocol-v2
Sec-WebSocket-Extensions: x-custom-extension
```

Primjer 4 - Odgovor servera pri uspostavljanju konekcije

HTTP Upgrade zahtijev se koristi kada klijent započinje konekciju običnim protokolom *HTTP* na portu 80, pa želi da se prebaci na neki drugi protokol. U tom slučaju, on definiše koji je novi protokol (npr. protokol *websocket*). Ukoliko server

odgovori pozitivno na *Upgrade* zahtijev, završava se rukovanje (engl. *handshake*) i može početi razmijena podataka.

Postoji nekoliko prednosti korišćenja *HTTP*-a prilikom uspostavljanja konekcije. Jedna je ta što čini *WebSocket*-e kompatibilnim sa postojećom *HTTP* infrastrukturom: *WebSocket* server može biti pokrenut na portovima 80 i 443, a često se dešava da su kod mnogih klijenata to jedini otvoreni portovi [5].

Pretraživač sam umeće Origin zaglavlje u zahtijev klijenta. Na osnovu njega server može da odbije zahtjev ukoliko nije u skladu sa CORS-om (enlg. *Cross-Origin Resource Sharing*) [5]. U okviru uspostavljanja konekcije opciono se vrši pregovaranje između klijenta i servera o potprotokolima (*Sec-WebSocket-Protocol*) i o proširenjima (*Sec-WebSocket-Extensions*). Server vraća koje potprotokole i proširenja podržava. Zaglavlje *Sec-WebSocket-Version* govori koju verziju *WebSocket* protokola klijent želi da koristi.

Neki od najčešćih scenarija ugrožavanja bezbijednosti komunikacije putem *WebSocket* protokola su:

- Onemogućavanje usluga (engl. *Denial of Service DoS*);
- Čovjek u sredini (engl. *Man in the middle*);
- XSS napad.

Kod DoS napada je akcenat na onemogućavanju usluga, tj. nedostupnost servera prilikom zahtijeva klijenata. Najčešće se realizuje tako što se šalje ogroman broj zahtijeva serveru gotovo istovremeno, zbog čega server prestaje da odgovara na zahtjeve ili odgovara presporo.

Čovjek u sredini je vrsta napada u okviru kojeg se između klijenta i servera infiltrira napadač. U slučaju da konekcija nije enkriptovana, on vrlo lako može da čita poruke i da se predstavlja lažno. U ovakvim slučajevima je riješenje korišćenje enkriptovane, sigurne konekcije (*WebSocket Secure* WSS).

XSS (engl. *Cross-site scripting*) je napad u kojem napadač šalje *HTML* ili JavaScript kod koji se izvršava na klijentu. Postoji veliki broj načina kako se ova vrsta napada izvodi, i kako se sprečava. U principu, najjednostavnije je odbaciti poruke koje sadrže JavaSkript ili *HTML*, ili ih prepravljati (npr. uklanjanje oznaka) tako da ne mogu da naprave štetu.

4 Potiskivana obavještenja

Dvadeseti vijek i početak novog milenijuma su sa sobom donijeli do sada neviđen napredak komunikacionih tehnologija. Komunikacija u realnom vremenu na velikim udaljenostima je naša svakodnevnica.

Razvoj komunikacionih tehnologija je pratio i razvoj samih uređaja - od telegrafa i glomaznih radio predajnika/prijemnika, preko satelitskih telefona, do prvih mobilnih telefona. Krajem prve decenije 21. vijeka, dolazi do razvoja nove generacije mobilnih uređaja – takozvanih pametnih telefona. Razlika između mobilnog telefona sa početka devedesetih godina prošlog vijeka i današnjih pametnih telefona bi mogla dovesti u pitanje i samo korišćenje termina mobilni telefon za savremene uređaje. Današnji pametni telefoni sadrže gomilu funkcija gdje je telefon samo jedna od njih, što ih čini pravim malim računarima.

Sve prethodno navedeno je nastalo kao rezultat ljudske želje da u brzom tempu savremenog života što manje vremena i energije troši na prikupljanje informacija i efikasnu komunikaciju. Danas više nije potrebno tražiti informacije, informacije sada mogu same da dolaze do korisnika. Takve poruke, koje dolaze bez posebnog zahtevanja, nazivamo **potiskivana obaveštenja**.

4.1 Osnove sistema

Potiskivana obavještenja (engl. *push notifications*) bi se najlakše mogla definisati kao poruke koje se pojavljuju na ekranu mobilnog telefona. Ovo je prilično slobodna definicija, iz razloga što potiskivana obavještenja nisu vezana isključivo za aplikacije za mobilne uređaje, već i za veb aplikacije, aplikacije na personalnim računarima i slično. Svi savremeni operativni sistemi za mobilne uređaje podržavaju potiskivana obavještenja. Potrebno je naglasiti da su ona vezana za samu aplikaciju a ne samo za uređaj, iako prvo uređaj prima obavještenje a nakon toga ga prosleđuje aplikaciji.

Klijent može da primi potiskivano obavještenje u bilo kom trenutku, pod uslovom da je povezan na internet. Ovo znači da aplikacija u tom trenutku ne mora uopšte biti aktivna, čak ni u pozadini.

Tipičan primjer korišćenja potiskivanih obavještenja bi bio vezan za rezultate sportskih utakmica. Prosječni korisnik želi da dobije informaciju samo onda kada dođe do promijene rezultata neke utakmice, i ne želi da provodi vrijeme pored ekrana gledajući u aplikaciju koja prati rezultate utakmica. Ovaj zahtjev se može ispuniti korišćenjem potiskivanih obavještenja, koja se šalju korisniku samo onda kada dođe do promijene rezultata.

Razlikuju se dva tipa potiskivanih obavještenja:

- Udaljena (*remote*) – ova obavještenja nastaju onda kada aplikacija dobije poruku od servera sa obavještenjem. U nastavku će najviše o njima biti riječi;

- Lokalna (*local*) – potiskivanih obavještenja koja nastaju na samom uređaju kada su zadovoljeni određeni kriterijumi. Ti kriterijumi mogu biti vezani za vrijeme i datum (recimo korisnik treba da ode kod ljekara u 17h, pa dobija potiskivano obavještenje o tome pola sata ranije), mogu biti zadovoljeni periodično (npr. na svakih 15 minuta se izvršava neki zadatak na uređaju i kao rezultat prikazuje obavještenje), mogu biti vezani za određene GPS koordinate na kojima se trenutno nalazi korisnik ili brzinu kojom se kreće itd. U principu, ona nastaju samo onda kada aplikacija sve raspoložive podatke ima na samom uređaju i sama pravi obavještenje bez inicijalizacije od strane nekog servera. Ova obavještenja najčešće korisniku izgledaju isto kao i udaljena, iako ona nisu u pravom smislu potiskivana zato što ne postoji server koji „gura“ podatke.

4.2 Arhitektura servisa

Arhitektura servisa za slanje potiskivanih obavještenja bi se mogla definisati kao klijent – server arhitektura, međutim, postoji bitan posrednik između njih a to je **servis operativnog sistema za rad sa potiskivanim obavještenjima *OSPNS*** (engl. *Operating system push notification service*). Svaki od mobilnih operativnih sistema ima sopstveni servis, međutim postoje riješenja koja mogu da rade istovremeno na više platformi, kao što je Guglov *FCM Firebase Cloud Messaging*.

Osim servisa koje obezbijeđuju kompanije kao što su Epl (engl. *Apple*), Majkrosoft ili Gugl, potrebni su i paketi za razvoj softvera, skraćeno *SDK* (engl. *Software Development Kit)*. Kao i servise, i *SDK*-ove obezbijeđuju kompanije koje održavaju mobilne platforme. Zahvaljujući *SDK*-ovima, implementacija potiskivanih obavještenja unutar mobilnih aplikacija je prilično jednostavna.

Svaka aplikacija koja želi da koristi potiskivana obavještenja mora prvo da se registruje na OSPNS. Nakon registracije, razvijaoci aplikacije dobijaju *API*-je (engl. *Application Programming Interface*) koji su najjčešće *REST API*-ji. Uz *API*, oni dobijaju i sertifikat ili *API* ključ koji se proslijeđuje *OSPNS*-u od strane aplikacionog servera da bi se isti autentifikovao na *OSPNS* prilikom slanja potiskivanog obavještenja. Nadalje, razvijaoci aplikacije implementiraju potiskivana obavještenja u samoj aplikaciji uključivanjem *SDK*-ova i pisanjem odgovarajućeg koda. Nakon implementacije aplikacija se stavlja u distribuciju.

Kada korisnik preuzme aplikaciju i pokrene je prvi put, ona se povezuje sa *OSPNS*-om i dobija nazad svoj token uređaja od *OSPNS*-a. Token uređaja nije samo adresa uređaja, već adresa same aplikacije na uređaju. Taj token se dalje proslijeđuje serveru koji šalje potiskivana obavještenja (najčešće je u pitanju aplikativni server), pa server sada zna za kojeg je korisnika aplikacije vezan koji token. Bitno je naglasiti da uvijek može doći do promijene tokena uređaja, i da je uvijek potrebno ažurirati podatke na serveru.

Gotovo svi servisi za slanje potiskivanih obavještenja funkcionišu po istom principu, ali se u dokumentaciji često služe različitim terminima. Tako je na primjer kod Guglovog servisa token uređaja (engl. *device token*) ili registracioni ID (engl. *Registration ID*), kod Epl-a je samo token, a kod Majkrosofta je identifikator *URI* kanala (engl. *Channel URI identifier*). Za podatke pomoću kojih se autentifikuje server prilikom komunikacije sa *OSPNS*-om se nekada koristi izraz *API* ključ, nekada sertifikat, nekada autentifikacioni token.

Primjer arhitekture sistema za slanje potiskivanih obavještenja je dat na slici 6. Osnovne komponente su:

1. Aplikacioni server (pošiljalac obavještenja) – naziva se još i provajder. On najčešće vodi evidenciju o tokenima svih uređaja na kojima se nalazi aplikacija koja prima potiskivana obavještenja i inicira slanje potiskivanog obavještenja ka *OSPNS*-u;
2. Servis za slanje potiskivanih obavještenja OSPNS – služi kao posrednik između servera i klijenta. Njemu se obraćaju i server (kod autentifikacije) i klijentska aplikacija (radi dobijanja tokena). Osim ove funkcije, on služi i za usmjeravanje potiskivanog obavještenja od aplikacionog servera (pošiljaoca) ka konkretnim uređajima, tj. aplikacijama na osnovu tokena;
3. Uređaj i klijentska aplikacija – uređaj prima potiskivano obavještenje od servisa za slanje potiskivanih obavještenja i prosljeđuje ga klijentskoj aplikaciji.

Slika 6 - Arhitektura sistema

Samo potiskivano obavještenje je najčešće *JSON* poruka sa definisanim vrijednostima polja unutar nje. Postoji priličan broj polja koja definišu izgled i ponašanje potiskivanog obavještenja na uređaju, o čemu će biti riječi kasnije. Kako se ne radi o opštem standardu već svaki operativni sistem za mobilne uređaje ima svoj sopstveni servis, imena polja i njihove vrijednosti zavise od konkretnog *OSPNS*-a i operativnog sistema uređaja.

Server koji inicira slanje potiskivanog obavještenja pravi jednu *JSON* poruku i šalje *HTTP* zahtjev ka *REST API*-ju servisa za slanje potiskivanih obavještenja. Ta poruka obavezno mora imati primaoca – token(e) uređaja kojima se šalje obavještenje. Server u zahtjevu mora poslati i svoj *API* ključ koji je dodijeljen aplikaciji prilikom registracije na *OSPNS*. Ukoliko ključ nije validan onemogućava se slanje.

Nakon što *OSPNS* primi *HTTP* zahtjev od servera, on na osnovu ključa identifikuje koji je server tj. aplikacija u pitanju, kao i koji su to uređaji kojima treba proslijediti obavještenje. Te uređaje *OSPNS* pronalazi na osnovu tokena uređaja.

U poglavlju 4.1 je naveden primjer korišćenja potiskivanih obavještenja kod aplikacije za praćenje rezultata sportskih utakmica. Klasični scenario za slanje jednog takvog obavještenja bi bio sledeći:

1. Korisnik ima instaliranu aplikaciju koja prati rezultate utakmica na svom uređaju i povezan je na internet;
2. Kada dođe do promijene rezultata, server uzima tokene onih uređaja kojima želi da pošalje obavještenje i proslijeđuje ga ka servisu za slanje potiskivanih obavještenja, zajedno sa sadržajem obavještenja. Uz svaki ovakav zahtjev server šalje i svoj *API* ključ na osnovu kojeg se predstavlja *OSPNS*-u;
3. *OSPNS* šalje obavještenje konkretnim uređajima na osnovu tokena;
4. Uređaj tumači sadržaj obavještenja i proslijeđuje ga dalje aplikaciji;
5. Aplikacija prikazuje potiskivano obavještenje na uređaju korisnika, najčešće uz zvučni signal.

4.3 Istorijat

Potiskivana obavještenja su relativno mlada tehnologija. U junu 2009. godine je pokrenut prvi servis od strane kompanije Epl. Njegov naziv je *Apple Push Notification service,* skraćeno APNs. Prvobitno je funkcionisao samo za mobilne uređaje sa operativnim sistemom iOS 3.0. Sa operativnim sistemom iOS 5.0 je dodat centar za obavještenja (engl. *notification centre*) gdje su se na jednom mjestu nalazila sva potiskivana obavještenja svih aplikacija koje se nalaze na uređaju a koje podržavaju obavještenja. Naknadno je dodata podrška za lokalne aplikacije na operativnom sistemu MAC OS X 10.7, a podrška za potiskivana obavještenja veb aplikacija u verziji operativnog sistema MAC OS X 10.9 preko pregledača veba Safari 7.0 [13].

Ubrzo nakon prezentacije Eplovog proizvoda uslijedio je i odgovor kompanije Gugl. Sredinom 2010. godine, predstavljen je servis *Cloud to Device Messaging* (C2DM) za operativni sistem Android 2.2. Dvije godine kasnije, kompanija Gugl je najavila da razvija naslednika C2DM servisa, i već od avgusta 2012. godine C2DM je proglašen prevaziđenim, da bi 2015. godine bio potpuno ugašen. Njegov naslednik je *Google Cloud Messaging* (GCM). Novi servis GCM se ne može smatrati samo novom verzijom C2DM zato što ova dva servisa nisu interoperabilna, te je bilo neophodno koristiti novi *SDK* i vršiti izmjene koda u samoj Android aplikaciji kao i u kodu serverske aplikacije. Glavna poboljšanja su veći broj parametara pri slanju poruka, mogućnost slanja podataka unutar potiskivanog obavještenja veličine do 4 KB i uklanjanje granica za broj poslatih poruka. GCM se oslanja na serverske *API*-je i *SDK*-ove koje održava Gugl. Savremena verzija GCM-a je *Firebase Cloud Messaging* (*FCM*) koji za razliku od GCM-a može da se koristi i kod operativnih sistema iOS i za slanje potiskivanih obavještenja veb aplikacijama.

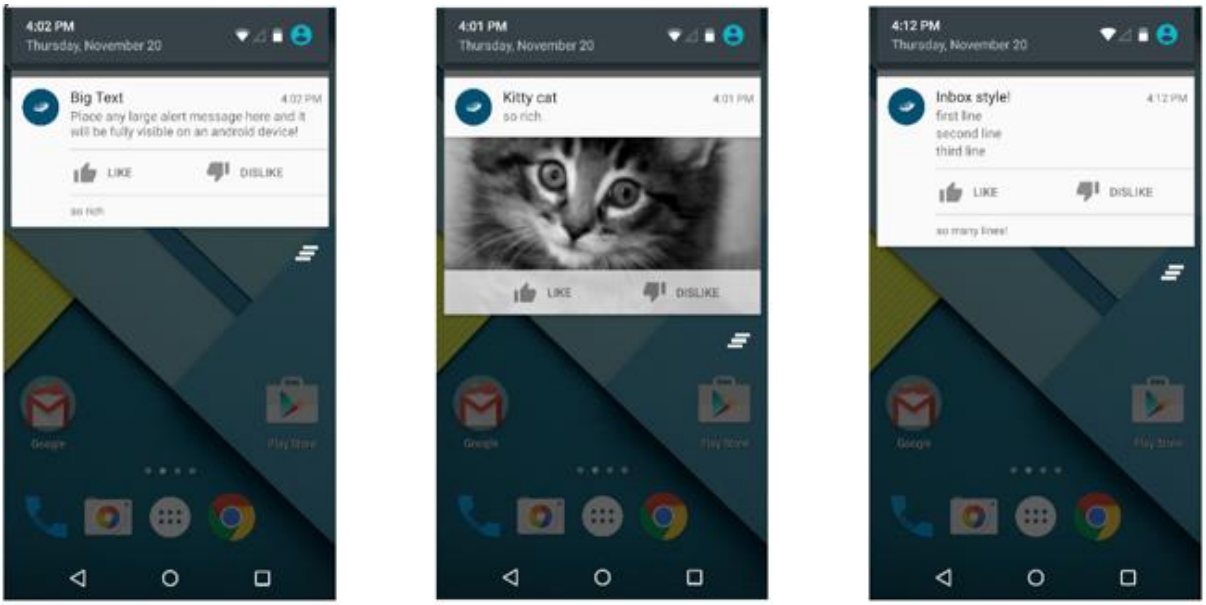
Operativni sistem Vindouz, razvijen od strane Majkrosofta takođe ima svoju verziju servisa za slanje potiskivanih obavještenja. Prvobitno se ovaj sistem zvao *Microsoft Push Notification Service* (MPNS) i bio je podržan na telefonima sa operativnim sistemom Vindouz Fon 8. Nije dugo trebalo da dobije svog naslednika u vidu *Windows Push Notification Service*-a (WPNS). Za razliku od svog prethodnika, WPNS cilja ne samo na mobilne već na sve uređaje na kojima je instaliran neki od Vindouz operativnih sistema, što podrazumijeva i uređaje kao što su *Xbox* ili personalni računari sa operativnim sistemom Vindouz 8, 8.1 ili 10.

4.4 Prikaz potiskivanih obavještenja

Svakako da prikaz potiskivanog obavještenja zavisi od konkretnog operativnog sistema, ali postoje tipovi prikaza koji su identični na gotovo svim operativnim sistemima. Osnovna tri tipa prikaza potiskivanih obavještenja su:

1. *Banner* – prikazuje se na vrhu ekrana, najčešće dok korisnik koristi telefon ali ne i aplikaciju čije je obavještenje, i nestaje nakon par sekundi;
2. *Lock screen* - prikazuje se na ekranu onda kada je telefon zaključan;
3. *Alert (dialog) box* – prikazuje se na sredini ekrana i blokira aplikaciju koja je trenutno na ekranu, tačnije blokira interfejs iza sebe.

Osim osnovnih potiskivanih obavještenja koja sadrže naslov i tekst, savremene verzije operativnih sistema podržavaju i bogata potiskivana obavještenja (engl. *rich push notifications*). Takva obavještenja omogućavaju dodavanje akcionih dugmića (pomoću kojih korisnik može pokrenuti neku akciju bez pokretanja aplikacije, već direktno iz obavještenja), zatim bogatiji sadržaj (osim teksta moguće je dodati fotografije, video snimke, linkove, dokumente itd.).



Slika 7 - Primjer centra za obavještenja na Android OS

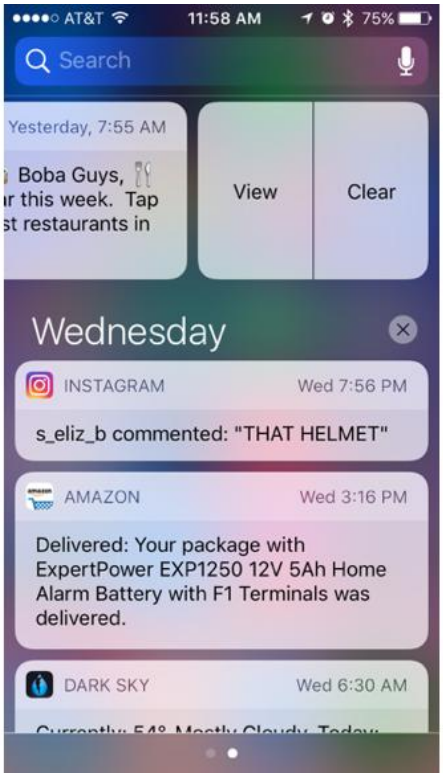
Prijem potiskivanog obavještenja uglavnom prati i zvučni signal, koji se može prilagoditi. Operativni sistemi grupišu obavještenja svih aplikacija na jedinstvenom

mjestu, pa korisnik može brzo da ih pregleda. Uglavnom se centru za obavještenja pristupa prevlačenjem prstom sa vrha ekrana ka dnu ekrana. Primjeri bogatih obavještenja sa slikom i akcionim dugmićima i centra za obavještena za operativne sisteme Android i iOS su dati na slikama 7 i 8.

4.5 Firebase Cloud Messaging - *FCM*

Do sada je uglavnom bilo riječi o uopštenim karakteristikama servisa za slanje potiskivanih obavještenja. U poglavlju Istorijat je pomenut konkretan servis koji je proizvod kompanije Gugl - *Firebase Cloud Messaging*. *FCM* je servis orijentisan i na veb i iOS platformu, a pritom je i besplatan. Osnovne mogućnosti ovog servisa su [14]:

- slanje potiskivanih obavještenja i poruka sa podacima aplikacijama – osim potiskivanih obavještenja moguće je slati i podatke koji se stavljaju na raspolaganje aplikaciji (*data* poruke) ;
- slanje obavještenja i poruka pojedinačnom uređaju, grupi uređaja ili uređajima koji su „pretplaćeni“ na određenu temu (engl. *subscribed to topic*) ;
- Slanje poruka od klijentske aplikacije ka serveru, pod uslovom da server podržava prošireni protokol za poruke i prisustvo *XMPP* (engl. *Extensible Messaging and Presence Protocol*).



Slika 8 - Primjer centra za obavještenja iOS uređaja

Slanje potiskivanih obavještenja korišćenjem *FCM* servisa je moguće na neki od sledećih načina:

1. Pravljenjem sopstvene aplikacije na serveru koja može da složi novo obavještenje, odredi primaoce i pošalje ga. Aplikacija može biti napravljena i korišćenjem paketa Admin *SDK* koji je napisan u jeziku Node.js;
2. Korišćenjem prilično intuitivne Guglove Konzole za obavještenja (engl. *Notification Console*) – veb aplikacije koja omogućava pravljenje i slanje obavještenja, analitiku, ciljanje publike itd.

4.5.1 Podešavanje klijenta

Da bi aplikacija bila u stanju da prima potiskivana obavještenja, potrebno je da ona obuhvati paket za razvoj softvera *FCM SDK*. Kod Androida, nakon uključivanja *SDK*-a unutar zavisnosti u **gradle.build** fajlu Android aplikacije, potrebno je definisati i dva Android servisa unutar manifest fajla:

1. Servis koji proširuje ***FirebaseMessagingService***. Ovaj servis je neophodan ukoliko aplikacija treba da radi bilo šta drugo osim pukog prijema obavještenja dok je aplikacija u pozadini. Ovo se odnosi na slanje poruka ka serveru, prijem obavještenja dok je aplikacija aktivna na ekranu itd. Najbitnije metode su ***onMessageReceived()*** koja se poziva prilikom prijema obavještenja i ***onMessageDeleted()***;
2. Servis koji proširuje ***MyFirebaseInstanceIDService***. Ovaj servis brine oko pravljena i ažuriranja registracionih ID-a tj. token*a*. Najbitnije metode su ***onTokenRefresh()*** koji se pokreće svaki put kada dođe do promijene tokena i ***getInstance().getToken()*** koji vraća trenutnu vrijednost tokena.

Prilikom prvog pokretanja aplikacije na uređaju, aplikacija dobija svoj token od strane *FCM* servera. Ovaj token se može promijeniti pod nekim okolnostima, a neke od njih su:

1. Korisnik je obrisao podatke aplikacije;
2. Korisnik je obrisao ili ponovo instalirao aplikaciju;
3. Aplikacija je prenijeta na novi uređaj.

U svakom trenutku je moguće pristupiti trenutnoj vrijednosti tokena. Svaki put kada dođe do promijene tokena, potrebno je voditi računa da se u toj situaciji poslednji token šalje serveru.

4.5.2 Prijem poruka na klijentu

Android aplikacije mogu da primaju tri vrste poruka od strane *FCM* servera

- Potiskivana obavještenja – poruka koja sadrži samo ključ ***notification*** i ostala polja vezana za izgled notifikacije (npr. naslov, sadržaj, ikonica itd.);
- Poruke sa podacima – poruka koja sadrži samo ključ ***data*** i proizvoljna polja unutar njega. Veličina poruke može biti do 4 KB;

- Hibrid dva prethodno navedena tipa poruka - istovremeno sadrži ***notification*** i ***data*** ključ (Primjer 5).

```
{
  "to" : "APa43bHu2n4M34egoK2t2KZ34FBaFUH-1RYqx...",
  "notification" : {
    "body" : "Sadržaj",
    "title" : "Naslov",
    "icon" : "ikonica"
  },
  "data" : {
    "ime" : "Marko",
    "prezime" : "Markovic",

    "jmbg" : "1103992331000"
  }
}
```

Primjer 5 - Hibridni tip *FCM* poruke

Od tipa poruke zavisi i način na koji će poruka biti obrađena u Android aplikaciji. Osim od tipa, obrada zavisi i od stanja aplikacije.

- Ukoliko je aplikacija u pozadini ili nije uopste pokretana, operativni sistem prikazuje obavještenje u gornjem dijelu ekrana i smiješta ga u centar za obavještenja. Ukoliko se ne radi o potiskivanom obavještenju nego o poruci sa podacima, podaci su raspoloživi u okviru metode ***FirebaseMessagingService.onMessageReceived()***. U slučaju hibridne poruke, podaci su raspoloživi nakon što korisnik klikom na obavještenje pokrene glavnu aktivnost kao dodaci (*intent extras*), a samo obavještenje prikazuje OS kao u slučaju čistog potiskivanog obavještenja. O aktivnostima, servisima, *intent*-ima će biti više riječi u poglavlju 5.2;
- Ukoliko je aplikacija aktivna i trenutno je na ekranu, sve poruke su raspoložive u okviru metode ***FirebaseMessagingService.onMessageReceived()*** bez obzira na njihov tip.

Da bi se obradila poruka, potrebno je pregaziti metod ***FirebaseMessagingService.onMessageReceived(RemoteMessage remoteMessage)***. Objekat tipa ***RemoteMessage*** sadrži pristupne metode za polja kao što su: *body, from, data* itd.

4.5.3 Podešavanje aplikacionog servera

FCM podržava slanje potiskivanih obavještenja od servera ka uređajima i slanje poruka u oba smjera, pod uslovom da server podržava *XMPP* protokol [14].

Za uspješno slanje potiskivanih obavještenja nisu potrebna dodatna podešavanja veb servera. Jedina potrebna stvar je da programski jeziku u kome je implementirana aplikacija za slanje potiskivanih obavještenja podržava slanje *HTTP POST* zahtjeva ka *FCM API*-jima. Korišćenjem paketa Admin *SDK* za serverske

aplikacije napisane u Node.js jeziku je olakšano upravljanje grupama uređaja i komponovanje poruka. Admin *SDK* je zvanični *Firebase* proizvod.

Ukoliko se vrši razmjena poruka od uređaja ka serveru, onda je na serveru potrebno omogućiti podršku za *XMPP* protokol. U nastavku će biti više riječi o *XMPP* protokolu.

4.5.4 Slanje poruka ka klijentima

Potiskivana obavještenja je moguće poslati pojedinačnom uređaju ili većem broju uređaja. Slanje pojedinačnom uređaju se vrši navođenjem tokena uređaja prilikom slanja zahtjeva. Slanje većem broju uređaja moguće na dva načina [14]:

1. Slanje uređajima koji su pretplaćeni na istu temu – u okviru *FCM*-a je moguće definisati teme na koje se aplikacije, odnosno korisnici, prijavljuju. Tako se za svaku temu vežu tokeni uređaja. Uređaj se pretplaćuje pozivom na ***FirebaseMessaging.getInstance(). subscribeToTopic("tema")***, pri čemu tema ne mora već postojati na serveru. U slučaju da ne postoji, biće automatski dodata i drugi uređaji će moći da se pretplaćuju na nju. Primjer *HTTP POST* zahtjeva ka *FCM* serveru je dat u primjeru 6.
2. Slanje grupi uređaja – najčešće se odnosi na slanje obavještenja većem broju uređaja koji pripadaju istom korisniku. Realizuje se tako što svaki od uređaja dobije svoj registracioni ID tj. token, a onda se ti tokeni pridruže jedinstvenom ključu koji je zajednički za sve uređaje iz te grupe. Prilikom odgovora, vraće se broj uspješno isporučenih obavještenja kao i neisporučenih, ukoliko ih ima. Moguće je upravljanje grupom (dodavanje novih uređaja, brisanje postojećih itd). Primjer *HTTP POST* zahtjeva ka *FCM* serveru je dat u primjeru 7.

```
https://fcm.googleapis.com/fcm/send
Content-Type:application/json
Authorization:key=AIzaSyZ-1u...0GBYzPu7Udno5aA
{
  "to": "/topics/sport",
  "data": {
    "message": "Test notifikacija za pretplaćene na sport",
  }
}
```

Primjer 6 - *HTTP* POST zahtjev za slanje poruke pretplaćenim uređajima

```
https://fcm.googleapis.com/fcm/send
Content-Type:application/json
Authorization:key=AIzaSyZ-1u...0GBYzPu7Udno5aA

{
  "to": "kljucGrupe",
```

```
  "data": {
     "message": "Test poruka grupi uređaja"
  }
}
```

Odgovor:

```
{
  "success":3,
  "failure":1,
  "failed_registration_ids":[
    "regId1"
  ]
}
```

Primjer 7 - *HTTP* POST zahtjev za slanje obavještenja grupi uređaja i *HTTP* odgovor

Pošto je već bilo riječi o tipovima poruka koje se razmijenjuju u okviru *FCM*-a, neophodno bi bilo i pomenuti neka od polja potiskivanog obavještenja koje je moguće prilagoditi:

- *time_to_live* – određuje životni vijek poruke. Poruke sa vrijednošću ovog polja jednakom 0 se odmah šalju. U slučaju neuspijeha nema ponovnog slanja. Za veće vrijednosti *FCM* pokušava slanje sve dok ne istekne zadati *time_to_live.* Maksimalna i podrazumijevana vrijednost je 28 dana (2 419 200 sekundi);
- *collapse_key* – postoje slučajevi kada nije potrebno slati sve poruke uređaju. Na primjer, ukoliko server šalje poruku za sinhronizaciju aplikaciji, dovoljno je poslati samo poslednju, pošto je ona najbitnija. Postoje i obrnuti slučajevi kada je potrebno poslati sve poruke (recimo kod aplikacija za razmijenu instant poruka). U tom slučaju vrijednost ovog polja je potrebno postaviti na *false;*
- *priority* – definiše važnost poruke. U slučaju da je prioritet poruke *normal*, poruke možda neće biti odmah isporučene uređaju ukoliko se uređaj aktivno ne koristi. Za prioritet *high*, poruke će biti odmah isporučene bez obzira da li je uređaj zaključan ili nije. Svrha ovog polja je da omogući efikasnije korišćenje baterije uređaja;
- *notification* – polje koje može sadržati polja poput *body, title, icon*;
- *data* – sadrži proizvoljna polja;
- *to* – sadrži token jednog ili grupe uređaja;
- *registration_ids* – niz tokena uređaja kojima se istovremeno šalju poruke.

Nakon što aplikacioni server pošalje poruku *FCM* servisu, poruka se smiješta u red za isporuku. U slučaju da je uređaj povezan na *FCM* i da je internet veza dobra, poruka se isporučuje gotovo odmah. U slučaju da je uređaj povezan na *FCM* ali nije aktivan, poruka dobija manji prioritet i smiješta se u red za isporuku. U slučaju da uređaj nije povezan, poruka stoji u redu najviše 28 dana. Ukoliko bude isporučena ili joj istekne polje *time_to_live*, uklanja se iz reda.

Ukoliko se uređaj povezao sa *FCM*-om nakon više od 28 dana ili ima više od 100 poruka koje čekaju u redu za isporuku uređaju, u aplikaciji se pokreće metod ***FirebaseMessagingService.onMessageDeleted()***. Cilj je da se kroz ovaj metod izvrši potpuna sinhronizacija između servera i klijentske aplikacije.

4.5.5 XMPP protokol i slanje poruka ka serveru

Prošireni protokol za poruke i prisustvo (*Extensible Messaging and Presence Protocol*, *XMPP*) je komunikacioni protokol koju služi za razmijenu manjih poruka i informacija o prisustvu. Zasnovan je na *XML*-u. Poznat je još pod imenom Džaber *(Jabber*). Uz pomoć ovog protokola moguće je uspostaviti trajnu asinhronu dvosmijernu vezu ka *FCM* serveru. Glavna prednost kod korišćenja ovog protokola je što nema potrebne da se stalno uspostavlja i raskida veza za svaki zahtjev, što dovodi do uštede u resursima. Druga bitna stvar je što je uz pomoć ovog protokola moguće slati poruke od uređaja ka *FCM* serveru. Moguće je koristiti XMPP i *HTTP* protokol paralelno.

Postoje dva *FCM XMPP* servera kojima je moguće pristupiti. Jedan je produkcioni, a drugi je testni. Nakon autentifikacije na *FCM* server, uspostavlja se konekcija i moguće je razmijenjivati poruke. Poruke su zapravo *JSON* niske omotane u *XML*-u. (Primjer 8)

```
<message id="">
  <gcm xmlns="google:mobile:data">
  {
    //identifikovanje aplikacije koja šalje poruku
    "category": "com.primjer.aplikacija",
    "data": {
      "test": "test test",
    },
    "message_id": "m-111",
    "from": "registracioni ID"
  }
  </gcm>
</message>
```

Primjer 8 - XMPP poruka koju server prima od klijenta

Za svaku ovakvu poruku aplikacioni server mora da vrat po ACK poruku ka *FCM* serveru, koja označava da je poruka uspješno primljena. Ukoliko *FCM* server ne dobije ACK poruku od aplikacionog servera, pokušaće da je pošalje opet. Osim ACK poruke, moguće je poslati i NACK poruku koja obavještava o neuspješnom prijemu (nepravilan *JSON*, nepravilan token uređaja i slično).

5 Operativni sistem Android

Android je softverska platforma za mobilne uređaje. Razvijen je od strane istoimene kompanije koja je funkcionisala pod pokroviteljstvom Gugla, sve dok je nije otkupio. Dostupan je od oktobra 2008. kao projekat otvorenog koda (pod *Apache* licencom). Zasnovan je na Linux jezgru, a implementira virtualnu mašinu Java (Dalvik), pregledač veba na bazi *WebKit*-a, bazu *SQL* i kompletan pristup hardveru uređaja. Ne samo da je danas jedan od najrasprostranjenijih operativnih sistema za tablet uređaje, mobilne telefone i sl., već se radi i o operativnom sistemu sa najvećim udjelom na globalnom računarskom tržištu. Kako se radi o prilagodljivom operativnom sistemu, vrlo lako se može implementirati na skupu raznovrsnih uređaja, poput televizora, pametnih satova, kamera.

Kako je Android projekat otvorenog koda postoji velika zajednica programera koja radi na njegovom razvijanju. Pošto se nalazi na velikom broju različitih uređaja, njegova implementacija je vremenski zahtijevna zato što je zvanična verzija prilagođena samo za Neksus uređaje, dok je za ostale potrebno vršiti izmjene.

5.1 Istorijat

Kompanija Andorid je osnovana 2003. godine u Palo Altu u Kaliforniji. Pvobitno je ciljala na tržište operativnih sistema za digitalne kamere, međutim polako se preusmjeravala na tržište operativnih sistema za pametne mobilne telefone. U avgustu 2005. godine Gugl je otkupio kompaniju Android, čime je ozvaničio svoj ulazak na tržište mobilnih telefona [15].

Konzorcijum tehnoloških kompanija *OHA* (engl. *Open Handset Alliance*), čiji je jedan od članova i Gugl, je osnovan 2007. godine u partnerstvu između proizvođača čipova, proizvođača uređaja i mobilnih operatera. Cilj ovog partnerstva je razvoj otvorenih standarda za mobilne uređaje. Zapravo, njihov prvi proizvod je Android. Gugl je kasnije lansirao svoju seriju Neksus uređaja.

Dvije interesatne činjenice su vezane za operativni sistem Android. Prva je da je sam Android prije svoje premijere bio zamišljen kao operativni sistem za uređaje sa fizičkom tastaturom. Tek naknadno je u specifikaciji dodat ekran na dodir [15]. Druga je vezana za nazive nadogradnji operativnog sistema – svaka nadogradnja nosi ime nekog dezerta (*Nougat*, *IceCream Sandwich, Jelly Bean* itd).

5.2 Osnovne karakteristike

Iako je zasnovan na Linuks jezgru, Android nije Linuks operativni sistem. Ne podržava standardni skup GNU biblioteka, ni sistem prozora, tako da nije moguće koristiti postojeće Linuks aplikacije. Aplikacije su pisane najčešće u programskom jeziku Java, mada je moguće koristiti programski jezik i C++. Ipak, nije moguće

izvršavanje Java programa napisanih npr. u J2ME, jer Android samo koristi sintasku Jave [16], a ne i identične biblioteke i virtualnu mašinu.

Do verzije Android 5.0, implementirana je Dalvik Java virtuelna mašina. Ova virtuelna mašina transformiše izvorni kod programa pomoću alata dk (koji je dio Android *SDK*-a) iz Java klasnih datoteka u .dex fajl. Međukod koji prevodi Dalvik nije Javin međukod, nego .dex (dex je skraćenica od *Dalvik EXecutable*). Transformacija formata omogućava bolju prilagođenost za rad na manjim procesorima boljim iskorišćavanjem resursa – u smislu memorije i procesora. Krajnji rezultat je da svaka Android aplikacija pokreće sopstvenu instancu virtuelne mašine Dalvik.

Od verzije 5.0 koristi se *Android Runtime* (*ART*), čime je potpuno zamijenjen Dalvik. Suština je da *ART* prevodi međukod u mašinski kod prilikom instalacije aplikacije. Time se štedi na resursima i omogućava veća efikasnost. Radi čuvanja kompatibilnosti unazad, i dalje se međukod distribuira u vidu .dex fajla.

Zahvaljujući centralnom repozitorijumu sa kojeg se preuzimaju aplikacije – *Google Play* prodavnici, bezbijednost na Android uređajima je na zavidnom nivou. Sadržaj svih Android aplikacija u prodavnici se skenira tako da je prilično teško zaraziti uređaj zlonamjernim softverom. Ranije verzije operativnog sistema Android su zahtijevale prilikom instalacije da korisnik odobori ili zabrani pristup sistemskim funkcijama (slanje poruka, poziva, praćenje lokacije, pristup internetu i slično), dok se od verzije 6.0 ove dozvole traže u toku izvršavanja koda. Aplikacija se preuzima kao .apk (*Android* PacKage) fajl, koji osim Java koda sadrži i resurse neophodne aplikaciji, kao što su slike, prevodi, zvukovi itd.

U srcu operativnog sistema Android se nalazi aplikacioni radni okvir (engl. *Application Framework*). Aplikacija se sastoji od sledećih komponenti: aktivnosti (engl. *Activities*), prijemnika emitovanja (engl. *Broadcast Receivers*), servisa (engl. *Services*) i snabdijevača sadržajem (engl*. Content Providers*) [17].

Aktivnost bi se kod Androida ugrubo mogao opisati kao pandan prozora ili forme na ekranu. Pokreće se preko *Intent*-a. *Intent*-i su mehanizam za pozivanje metoda, proslijeđivanje podataka i primanje rezultata. Na primjer, da bi se iz aplikacije otvorio *Activity* koji prikazuje sadržaj na nekom *URL*-u, dovoljno je napraviti *Intent*, njemu dodati potreban parametar a to je *URL* adresa i proslijediti taj *Intent* metodi ***startActivity()*** koja ga pokreće.

Servis je komponenta aplikacije koja se nalazi u pozadini, nema svoj grafički interfejs i uglavnom nema interakciju sa korisnikom (npr. detektor GPS lokacije u pozadini).

Android posjeduje zanimljiv mehanizam prilikom sistemskih događaja kao što su dolazna SMS poruka, dolazni poziv, uspješno pokretanje (engl. *boot*) uređaja itd. U tim situacijama operativni sistem šalje javno emitovanje (engl. *broadcast*) svim aplikacijama ii servisima, a samo one aplikacije koje imaju prijavljene prijemnike javnih emitovanja (engl. *broadcast receiver*) za određeni događaj (npr. dolazna SMS poruka) mogu da ga prime i odgovore na njega. Snabdijevači sadržaja proslijeđuju i održavaju trajne podatke u jednostavnim tabelama u relacionoj bazi podataka.

Interesantna stvar je način na koji Android upravlja aplikacijama, tako što ih ređa na steku i to na sledeći način: u slučaju da korisnik ne ugasi aplikaciju a pokrene neku drugu, ona dolazi na stek iznad prve aplikacije. Ako korisnik dodirne dugme za povratak unazad, menadžer aktivnosti vraća prvu aplikaciju na ekran. Takođe, životni ciklus aplikacije nije strogo kontrolisan, već njime upravlja sam operativni sistem na osnovu različitih parametara, od kojih su neki: koliko je već aplikacija aktivna, koliko ima raspoložive memorije na uređaju i slično.

5.3 Razvojno okruženje

Aplikacije za Android uređaje se uglavnom pišu na programskom jeziku Java, pri čemu mogu biti upotrebljavani i programski jezici C/C++.

Prva verzija razvojnog okruženja za razvoj Android aplikacija je bila *Eclipse IDE*, pri čemu je bilo potrebno instalirati dodatak (engl. *plugin*) razvojnih alata za Android ADT (engl. *Android development tools*). Krajem 2014. godine, kompanija Gugl je objavila Android Studio, koje postaje zvanično razvojno okruženje. Android Studio nije zasnovan na *Eclipse* alatu, već na *IntelliJ IDEA*.

Android *SDK*, koji je već pominjan, u sebi sadrži određene razvojne alate, kao što su: *debugger*, softverske biblioteke, emulator Android uređaja, dokumentaciju, primjere koda i lekcije za učenje.

Dosta je jednostavan za upotrebu, omogućava jednostavnu integraciju sa Gugl servisima poput *FCM*-a, međutim, prilično je zahtijevan sa stanovišta resursa: verzija 2.3.3 zahtijeva minimum tri gigabajta radne memorije, dok je preporučeno 8 gigabajta. Zbog ovakvih zahtijeva na računarima sa 4 do 6 gigabajta radne memorije Android Studio radi izuzetno sporo.

6 Aplikacija KlikTaxi

U manjim gradovima sa nedovoljno razvijenim gradskim saobraćajem, prisutan je trend porasta broja taksi vozila/udruženja. Pamćenje brojeva svih taksi udruženja može biti frustrajuće za korisnika, kao i pozivanje svih tih brojeva u krug u periodu najveće gužve sve dok se ne dobije slobodno vozilo. Istovremeno, ukoliko se koristi tradicionalni način naručivanja vozila putem telefonskog poziva ili eventualno SMS poruke, to znači i dodatne troškove za korisnika. Nije potrebno pominjati i vrijeme koje korisnik gubi dok nađe broj telefona udruženja, unese taj broj, saopšti adresu dispečeru i dobije odgovor.

Primarna namjena aplikacije KlikTaxi je da korisnicima olakša proces naručivanja taksi vozila. Kompletan sistem se sastoji od tri osnovne aplikacije:

- Korisničke mobilne aplikacije za operativni sistem Android;
- Dispečerske veb aplikacije;
- Administratorske veb aplikacije.

U nastavku rada će biti data logička arhitektura sistema i dati slučajevi korišćenja za korisničku mobilnu aplikaciju.

6.1 Logička arhitektura

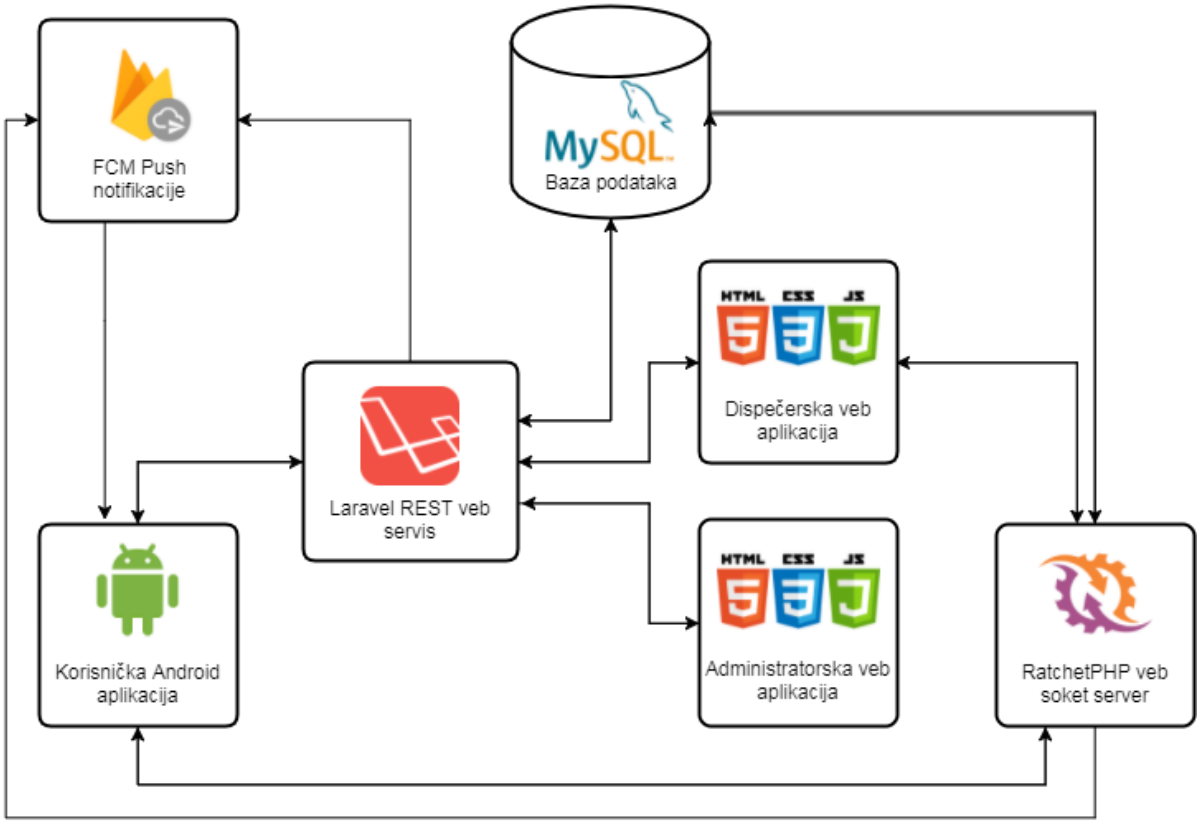
Već je naznačeno da se kompletan sistem sastoji od 3 osnovne aplikacije: korisničke, dispečerske i administratorske aplikacije. Za početak treba prepoznati tri vrste korisnika aplikacije:

- **Korisnik** - predstavlja sve one koji preuzmu korisničku Android aplikaciju i otvore nalog na KlikTaxi serveru;
- **Dispečer** - predstavlja dispečere zaposlene u taksi udruženjima, koji služe kao posrednici između taksi vozača i korisnika mobilne aplikacije;
- **Administrator** – predstavlja osobe koje održavaju KlikTaxi sistem, dodaju/ ažuriraju/ brišu podatke o udruženjima i dispečerima, upravljaju nalozima korisnika mobilne aplikacije, šalju obavještenja taksi udruženjima tj. dispečerima i sl.

Dijagram arhitekture KlikTaxi aplikacije je dat na slici 9. Korisnička mobilna aplikacija komunicira sa *REST* veb servisom, npr. prilikom prijave korisnika na aplikaciju, izmijene korisničkih podataka, otvaranja novog korisničkog naloga, pregleda svih ostvarenih vožnji i slično. Komunikacija u realnom vremenu između dispečerske veb aplikacije i korisničke Android aplikacije se uglavnom odvija putem *WebSocket*-a i/ili potiskivanih obavještenja. Zahvaljujući tome moguća je istovremena dvosmijerna komunikacija i dobre performanse s obzirom da korisnik i dispečer gotovo trenutno razmijenjuju poruke. Dispečerska veb aplikacija koristi i veb servis, u slučaju kada dispečer želi da pogleda podatke o prethodnim vožnjama i ukoliko ima privilegije izmijeni određene podatke.

U određenim situacijama, razmijena podataka između dispečerske i korisničke aplikacije se može vršiti korišćenjem potiskivanih obavještenja. Dispečerska aplikacija može putem *WebSocket*-a direktno da pristupi *API*-ju potiskivanih obavještenja i pošalje obavještenje korisničkoj aplikaciji. Istu mogućnost ima i administratorska aplikacija ali putem veb servisa, umjesto putem *WebSocket*-a.

U bazi podataka su pohranjeni svi podaci o korisnicima, dispečerima, vožnjama, taksi udruženjima i slično.



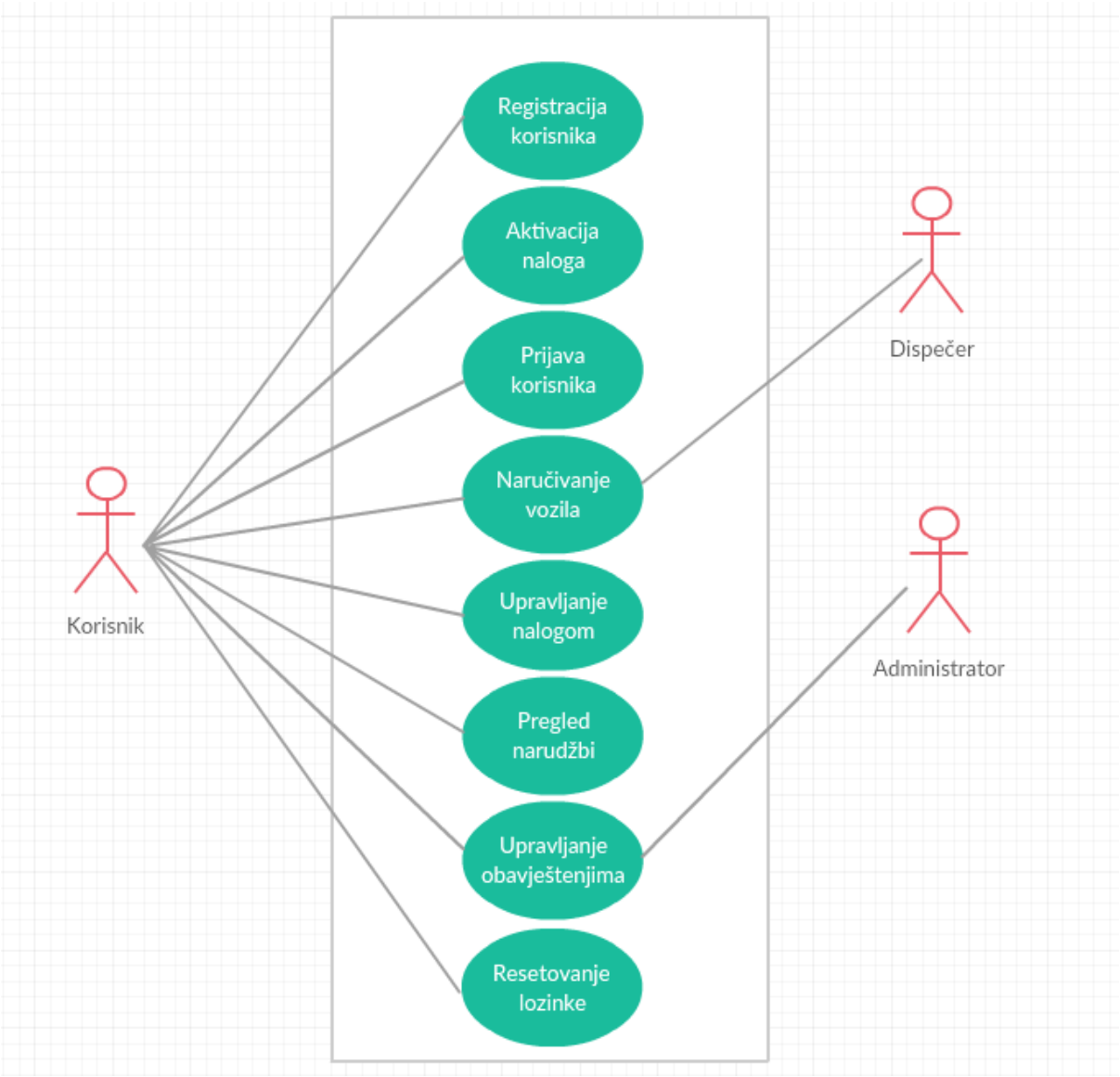
Slika 9 - Arhitektura aplikacije KlikTaxi

6.2 Slučajevi korišćenja mobilne aplikacije

U ovom poglavlju su obrađeni reprezentativni slučajevi korišćenja klijentske mobilne aplikacije. Reprezentativni slučajevi korišćenja su oni slučajevi korišćenja koji demonstriraju korišćenje tehnologija obrađenih u radu. Svaki slučaj potrebe je opisan osnovnim i alternativnim tokom.. Glavni dijagram slučajeva korišćenja je dat na slici 10.

Da bi korisnik koristio mobilnu aplikaciju potrebno je da prvo otvori nalog na KlikTaxi serveru (otvaranje naloga, tj. registracija korisnika, se odvija preko Android aplikacije). Nakon otvaranja naloga, potrebno je aktivirati isti. Aktivacija se vrši unošenjem broja koji korisnik dobija SMS-om. Nakon aktivacije naloga korisnik može da vrši naručivanje vozila, upravlja podacima svog naloga, pregleda istorijat vožnji i pregleda obavještenja. U slučaju da korisnik zaboravi lozinku, na zahtijev može da

dobije novu. Postoje još neki jednostavni slučajevi korišćenja, ali kako kod njih ne postoji potreba za bilo kakvom komunikacijom sa serverom, nisu obrađeni u ovom radu.



Slika 10 - Dijagram slučajeva korišćenja

6.2.1 Slučaj korišćenja – Registracija korisnika

Slučaj korišćenja: Registracija korisnika – otvaranje novog korisničkog naloga

Akter: Korisnik, osoba koja želi putem aplikacije da poziva taksi vozila

Preduslov: Korisnik je preuzeo aplikaciju sa Gugl prodavnice i odabrao dugme za registraciju

Osnovni scenario:

1. Korisnik unosi neophodne podatke za registraciju (ime i prezime, broj telefona, lozinku i pol i klikom na dugme “Registruj se” proslijeđuje podatke serveru
2. Korisnik je uspješno registrovan i prikazuje se forma za aktivaciju naloga. Podaci o nalogu koje je korisnik unio se čuvaju na uređaju

Alternativni scenario:

2.1. Korisnik nije unio sve podatke i ne može da se registruje

2.2. Korisnik je unio lozinku kraću od osam karaktera

2.3. Korisnik nije unio odgovarajući format broja telefona

2.4. Lozinke se ne podudaraju

2.5. Korisnik nije prihvatio uslove korišćenja

2.6. Broj telefona koji je unio korisnik već postoji u KlikTaxi bazi i registracija nije moguća

Korisnik o svakom slučaju alternativnog scenarija dobija odgovarajuću poruku.

6.2.2 Slučaj korišćenja – Aktivacija naloga

Slučaj korišćenja: Aktivacija korisničkog naloga – verifikovanje broja telefona

Akter: Korisnik, osoba koja želi da putem aplikacije poziva taksi vozila

Preduslov: Korisnik je otvorio nalog čiji je status 0 - “neaktivan” i dobio SMS poruku sa kodom za aktivaciju naloga

Osnovni scenario:

1. Korisnik dobija SMS sa kodom za aktivaciju
2. Aplikacija “presrijeće” dobijeni SMS i čita aktivacioni kod iz njega. Kod se automatski proslijeđuje serveru.
3. Server obaviještava korisnika da je unijeti kod validan i preusmjerava ga na glavni ekran za naručivanje taksi vozila

Alternativni scenario:

1.1. Korisnik nije dobio kod za aktivaciju, i klikom na dugme resetuj pin traži novi kod

2.1. Aplikacija nije bila u mogućnosti da automatski pročita i pošalje kod, pa ga korisnik unosi ručno i klikom na dugme “Pošalji pin” šalje serveru

3.1. Pin nije ispravan i korisnik dobija odgovarajuću poruku

3.2. Korisnik nije unio pin o čemu dobija odgovarajuću poruku

6.2.3 Slučaj korišćenja – Prijava korisnika

Slučaj korišćenja: Prijava korisnika na aplikaciju

Akteri: Korisnik, osoba koja želi da putem aplikacije poziva taksi vozila

Dispečer, osoba koja odgovara na zahtijeve korisnika

Preduslov: Korisnik posjeduje nalog na serveru

Osnovni scenario:

1. Korisnik unosi neophodne podatke za prijavu na aplikaciju (broj telefona i lozinka). Klikom na dugme “Prijavi se” šalje podatke serveru
2. Podaci su validni i korisnik je uspiješno prijavljen na sistem. Podaci o nalogu dobijeni u odgovoru servera se čuvaju na uređaju

Alternativni scenario:

1.1. Korisnik nije unio podatke
2.1. Korisnik je unio netačne podatke
2.2. Korisnik je unio tačne podatke ali nalog nije aktiviran i preusmijerava se na formu za aktivaciju naloga
2.3. Korisnik je unio tačne podatke ali je njegovom nalogu zabranjen pristup aplikaciji. Korisnik dobija poruku o razlogu zabrane pristupa.

6.2.4 Slučaj korišćenja – Resetovanje lozinke

Slučaj korišćenja: Resetovanje lozinke

Akter: Korisnik koji je zaboravio lozinku i traži novu

Preduslov: Korisnik posjeduje nalog na serveru

Osnovni scenario:

1. Korisnik pokreće aplikaciju i klikom na dugme “Zaboravili ste lozinku ? Kliknite ovdje!” otvara formu za reset lozinke
2. Korisnik unosi svoj broj telefona koji je vezan za nalog za koji da resetuje lozinku i klikom na dugme “Resetuj lozinku” šalje zahtijev serveru
3. Ukoliko je broj telefona vezan za neki nalog, korisnik dobija poruku da je poslat SMS/e-mail (u zavisnosti šta je korisnik unio od podataka) sa *URL*-om koji je potrebno otvoriti da bi se dobila nova lozinka
4. Korisnik otvara *URL* iz SMS-a i dobija novu lozinku

Alternativni scenario:

3.1. Broj telefona nije vezan ni za jedan nalog o čemu korisnik dobija poruku

4.1. Korisnik nije dobio SMS/e-mail (unio je broj tuđeg telefona ili je došlo do problema u radu SMS ili e-mail servisa)

6.2.5 Slučaj korišćenja – Naručivanje vozila

Slučaj korišćenja: Naručivanje taksi vozila

Akter: Korisnik koji poziva taksi vozilo putem aplikacije

Dispečer, zaposleni u taksi službi koji odgovara na zahtijeve korisnika

Preduslov: Korisnik posjeduje nalog na serveru koji je aktiviran i nije mu zabranjen pristup, i sa tim nalogom je prijavljen na aplikaciju

Osnovni scenario:

1. Korisnik je uspješno uspostavio vezu sa serverom (dispečerima) i dobio spisak svih aktivnih dispečera tj. taksi udruženja. Otključavaju se polja za unos podataka
2. Aplikacija sama detektuje u kom gradu se nalazi korisnik i odabira taj grad.
3. Korisnik unosi adresu na kojoj želi da dobije vozilo
4. Korisnik bira udruženje od koga želi da naruči vozilo i klikom na dugme “Naruči” šalje zahtijev serveru (dispečerima)
5. Zahtijev je uspješno dostavljen dispečerima i korisnik dobija poruku o tome. Takođe, prikazuje se vrijeme čekanja na odgovor
6. Dispečer korisniku odobrava vožnju i vožnji je dodijeljen broj vozila i broj minuta koji je potreban da vozilo dođe na odabranu adresu. Prikazuje se ekran sa podacima o narudžbi. U slučaju da je aplikacija pokrenuta u pozadini prikazuje se potiskivano obavještenje
7. Korisnik može da prati kretanje vozila klikom na dugme “Prati”
8. Korisnik potvrđuje da je vozilo došlo na odabranu adresu

Alternativni scenario:

1.1. Korisnik nije uspostavio vezu sa dispečerima i polja za unos podataka ostaju zaključana
1.2. Korisnik je uspostavio vezu ali nema aktivnih dispečera
1.3. Korisnik je uspostavio vezu ali ima narudžbu kojoj nije definisan status (nije još odgovoreno na nju ili nije potvrdio da je vožnja završena) i prikazuje mu se ekran sa podacima o naružbi
2.1. Aplikacija nema dozvolu za pristup lokaciji korisnika, pa korisnik mora ručno da odabere grad
1.1. Korisnik klikom na čiodu pored automatski detektuje svoju lokaciju preko sa Gugl mape. U mapi može da pomijera čiodu dok ne dobije tačnu tekstualnu adresu. Ta adresa se nakon toga unosi u polje za adresu na glavnom ekranu (Slika 16)

1.2. Korisnik klikom na jednu od omiljenih lokacija automatski popunjava polje za adresu i grad

4.1. Korisnik ne želi da sam bira udruženje već označava polje “Automatsko biranje”. Sada se udruženja naručuju lančano sa liste, sve dok se ne dobije slobodno vozilo ili ne dođe do kraja liste. Redosled ove liste je moguće podešavati klikom na “podesi redosled”

6.1. Korisniku nije odobrena vožnja jer nema slobodnih vozila, o čemu korisnik dobija poruku ili potiskivano obavještenje

6.2. Korisniku nije odobrena vožnja jer unijeta adresa nije dovoljno jasna, o čemu korisnik dobija poruku ili potiskivano obavještenje

7.1. Korisnik ne može pratiti vozilo pošto se podaci o tom vozilu ne prate u bazi podataka

8.1. Korisnik prijavljuje kašnjene vozila

8.2. Korisnik otkazuje vožnju

6.2.6 Slučaj korišćenja – Upravljanje nalogom

Slučaj korišćenja: Upravljanje nalogom

Akter: Korisnik koji ima otvoren nalog na KlikTaxi serveru

Preduslov: Korisnik posjeduje nalog na serveru i prijavljen je sa njim na KlikTaxi apllikaciju

Osnovni scenario:

1. Korisnik bira stavku “Moj nalog” u meniju i otvara formu Moj nalog, koja je popunjena lokalno snimljenim podacima o nalogu
2. Korisnik unosi podatke koje želi da promijeni (ime i prezime, e-mail, lozinku, datum rođenja i pol i klikom na dugme “Sačuvaj” prosljeđuje podatke serveru
3. Korisnik je uspješno sačuvao podatke i prikazuje se odgovarajuća poruka

Alternativni scenario:

3.1. Korisnik je unio lozinku kraću od osam karaktera

3.2. Lozinke se ne podudaraju

Korisnik o svakom slučaju alternativnog scenarija dobija odgovarajuću poruku.

6.2.7 Slučaj korišćenja – Pregled narudžbi

Slučaj korišćenja: Pregled narudžbi

Akter: Korisnik koji ima otvoren nalog na KlikTaxi serveru

Preduslov: Korisnik posjeduje nalog na serveru i prijavljen je sa njim na KlikTaxi apllikaciju

Osnovni scenario:

1. Korisnik bira stavku “Istorijat” u meniju
2. Korisnik od servera dobija spisak svih narudžbi. Narudžbe i detalji vezani za njih se prikazuju u listi.
3. Klikom na dugme “Obriši” korisnik može da obriše spisak svih narudžbi

Alternativni scenario:

2.1. Korisnik uopšte nema narudžbi, pa je prikaz nepromijenjen. Korisnik dobija odgovarajuću poruku

6.2.8 Slučaj korišćenja – Upravljanje obavještenjima

Akteri: Korisnik koji ima otvoren nalog na KlikTaxi serveru

Administrator, održava KlikTaxi sistem

Preduslov: Korisnik posjeduje nalog na serveru i prijavljen je sa njim na KlikTaxi aplikaciju

Osnovni scenario:

1. Administrator unosi novo obavještenje
2. Obavještenje je unijeto i korisnik dobija poruku o tome
3. Korisnik otvara poruku i gleda spisak obavještenja

7 Implementacija aplikacije

Mobilna aplikacija predstavlja korisničku aplikaciju (za korisnika se smatra osoba koja koristi klijentsku KlikTaxi Android aplikaciju). U prethodnom poglavlju je obrađena arhitektura i slučajevi korišćenja, dok će u nastavku biti riječi o implementaciji slučajeva korišćenja sistema. Cilj ovog poglavlja je da se kroz konkretnu implementaciju aplikacije prikaže korišćenje tehnologija opisanih u prethodnim poglavljima. U nastavku poglavlja će biti dati i primjeri poruka koje se razmijenjuju između korisnika i servera.

Klijentski dio mobilne aplikacije je implementiran u jeziku Java korišćenjem paketa Android *SDK*. Najstarija verzija operativnog sistema Android na kojoj aplikacija može biti instalirana je 4.2.

7.1 Biblioteke potrebne za implementaciju veb servisa, potiskivanih obavještenja i WebSocket-a u Android aplikaciji

Da bi se mogli koristiti veb servisi, potrebno je da postoji mogućnost slanja *HTTP* zahtjeva ka veb servisu sa klijentske strane. Kod Androida, ovu mogućnost daje biblioteka *HttpURLConnection*. Korišćenjem ove biblioteke je moguće poslati različite vrste *HTTP* zahtjeva ka odabranom serveru. *HttpURLConnection* klasa ima dosta polja čije se vrijednosti mogu mijenjati. *HttpURLConnection* objekat koji služi za slanje *POST* zahtjeva ka serveru sa parametrima kljuc - vrijednost je dat na primjeru 9.

Radi jednostavnosti, u aplikaciji je napisana nova klasa čiji konstruktor kao argumente uzima *URI* resursa i oznaku *HTTP* metoda, kao i proizvoljan niz parametara i njihovih vrijednosti. Ta klasa za svaki zahtjev ka *REST* servisu pravi *HttpURLConnection* objekat.

```
try {
  StringBuilder paramData = new StringBuilder();
  paramData.append("kljuc","UTF-8"));
  paramData.append('=');
  paramData.append("vrijednost","UTF-8"));
  byte[] pdBytes = paramData.toString().getBytes("UTF-8");
  URL link=new URL(("adresa.servera");
  HttpURLConnection conn = (HttpURLConnection) link.openConnection();
  conn.setRequestMethod("POST");
  conn.setRequestProperty("Content-Type", "application/x-www-form-
URLencoded");
  conn.setRequestProperty("ContentLength",
String.valueOf(pdBytes.length));
  conn.setDoOutput(true);
  conn.getOutputStream().write(paramDataBytes);
  is = conn.getInputStream();
}
```

Primjer 9 - Slanje *HTTP POST* zahtjeva korišćenjem *HttpURLConnection*

Android nema u sebi ugrađen *API* za korišćenje *WebSocket*-a. Ipak, postoji bilioteka *JavaWebSocket* koja omogućava korišćenje *WebSocket*-a u skladu sa istim standardom protokola koji je podržan od strane pregledača veba, pa je i njena upotreba jednostavna. Svaka poruka koja se razmijenjuje putem *WebSocket*-a treba sa sobom da nosi informaciju o kojoj vrsti akcije se radi (prijava korisnika na soket, odjava korisnika, naručivanje nove vožnje itd), što je u KlikTaxi aplikaciji realizovano na sledeći način: uz svaku *JSON* poruku se dodaje posebno polje sa nazivom **kod**. Polje **kod** je trocifreni broj koji označava o kojem događaju je riječ. Primjeri nekih od kodova su:

- 101 - zahtjev za autentifikaciju dispečera;
- 102 - zahtjev za odjavu dispečera;
- 301 - prijava na soket;
- 302 - odjava sa soketa;

Programer mora sam da definiše šta je značenje kojeg koda i šta od sadržaja treba da se nalazi u poruci koja sadrži taj kod. Zahvaljući ovome, dobija se nešto nalik *RPC* arhitekturi, pri čemu mora da postoji dogovor o sadržaju poruka koje se razmijenjuju.

Za korišćenje potiskivanih obavještenja dovoljno je u Android projekat uključiti *FCM SDK*, koji je već opisan u poglavlju 4.5.

Sve poruke koje se razmijenjuju između klijenata i servera su u *JSON* formatu, bez obzira na korišćenu tehniku povezivanja.

7.2 Korišćene tehnologije

Za implementaciju logičkih komponentni arhitekture sistema korišćeno je nekoliko tehnologija. Postoji veliki broj tehnologija za svaku od komponenti sistema sa donekle sličnim karakteristikama i performansama, tako da ovaj spisak možda ne predstavlja najbolje od najboljih, jer su tehnologije birane spram preferenci autora rada.

Svi podaci su pohranjeni u bazi podataka *MySQL*. Za razvoj klijentskog dijela administratorske i dispečerske veb aplikacije je korišćen *HTML*5, *CSS* i uglavnom čisti *JavaScript* uz rijetko korišćenje *jQuery* biblioteke. Klijentski dio korisničke aplikacije je napisan u programskom jeziku Java korišćenjem paketa Android *SDK*.

Za razvoj veb servisa korišćen je *Laravel*, a za razvoj *WebSocket* servera *Ratchet PHP*. Obje tehnologije počivaju na *PHP*-u. *PHP* (engl. Hypertext Preprocessor) je serverski skriptni programski jezik otvorenog koda za dinamičko generisanje *HTML* dokumenata. *PHP* je skriptni jezik pomoću kojeg se pravi *HTML* dokument. On se korišćenjem *PHP* skripte popunjava dinamičkim sadržajem i isporučuje klijentu nakon popunjavanja. Skripta je nevidljiva klijentu, njemu je vidljiv isključivo *HTML* kod.

7.2.1 Laravel

Za realizaciju veb servisa je korišćen razvojni okvir *Laravel*, zasnovan na jeziku *PHP*. *Laravel* je besplatni razvojni okvir otvorenog koda. U principu se koristi za pravljenje veb aplikacija zasnovanih na arhitekturi *MVC* (engl. *Model View Controller*). Arhitektura *MVC* je danas veoma zastupljena. Razlikuje tri sloja: sloj modela, sloj pogleda i sloj kontrolera. Primarni cilj ovakve arhitekture je da razdvoji prikaz podataka od interakcije korisnika sa podacima. U sloju pogleda korisnik dobija prikaz stanja modela i može da poziva određene operacije koje se izvršavaju nad modelom. Kontroler služi kao posrednik između pogleda i modela. Putem pogleda on dobija informaciju o operaciji nad modelom koju korisnik želi da izvrši, i prosleđuje tu informaciju modelu. Ukoliko se stanje modela promijeni, kontroler o tome obavještava pogled, pa tako korisnik vidi novo stanje modela.

Neki od osnovnih koncepta koje uvodi Laravel-a su:

- Elovkentni ORM (engl. *Eloquent object relational mapping ORM*) koji u praksi preslikava tabele u bazi podataka kao klase, a instance klasa predstavljaju jedan slog u tabeli;
- Graditelj upita (engl. *Query Builder*) koji omogućava da se ne moraju direktno pisati *SQL* upiti, već postoji skup klasa i metoda putem kojih mogu da se vrše upiti;
- Aplikaciona logika se može implementirati korišćenjem kontrolera ili kao dio deklaracije putanja (engl. *route*) ;
- *REST*-oliki kontroleri omogućavaju da se razdvoji logika između *HTTP GET* i *POST* zahtjeva;
- Automatsko učitavanje klasa, pa nije potrebno ručno ubacivanje putanja u kod;
- *Blade templating engine* koji kombinuje jedan ili više šablona (engl. *template*) sa modelom podataka i formira konačni pogled. Posjeduje sopstvene programske kontrolne strukture (uslovi, petlje) koje se kasnije mapiraju u *PHP*-ove strukture;
- Migracije – omogućavaju lakše verzioniranje. Putem migracija je moguće mapirati koje su izmjene u bazi podataka bile u kojoj verziji aplikacije, što omogućava koherentnost između aplikacije i baze podataka;
- Punjenje baze podataka (eng. *Database seeding)* omogućava da se prilikom testiranja aplikacije baza puni odabranim podacima ili da se prilikom prve instalacije pune samo potrebni podaci;
- Automatizovano testiranje;
- Automatska paginacija strana.

Još jedna od karakteristika Laravel-a jeste komandni interfejs *Artisan*. Putem njega se mogu izvršavati komande punjenja baze, migracija, generisana zajedničkog koda kontrolera i slično.

7.2.2 Ratchet*PHP*

Za realizaciju *WebSocket* servera korišćen je *RatchetPHP* server zasnovan na *PHP*-u. *PHP* se uglavnom koristi za izvršavanje skripti sa kratkim životnim vijekom. Prilikom zahtjeva klijenta prema serveru, server pokreće *PHP* skriptu, izvršava je i šalje nazad *HTML* dokument ka klijentu. U slučaju *WebSocket*-a skripta se pokreće preko *PHP*-ovog komandnog interfejsa (*PHP* CLI) i ostaje aktivna svo vrijeme dok radi server [18].

Jezgro *Ratchet*-a se sastoji od komponenti. Svaka komponenta ima svoj *ComponentInterface* preko koga se može pristupati metodama komponente. Jedna od bitnijih komponenti je *WAMP* (engl. *WebSocket Application Messaging Protocol*) koja omogućava lakšu interakciju između *JavaScript* klijenta i *Ratchet* servera. Ova komponenta sadrži metode preko kojih su mogući *RPC* pozivi (*onCall()* metod) i šabloni objavi (engl. *publish*) i pretplati se (engl. *subscribe*). Ovi šabloni rade po sledećem principu: svaki korisnik se pretplati za određenu temu (npr. pretplati se samo za sadržaj iz politike, metoda *subscribe()*). Preko metode *publish()* se na serveru objavljuje određeni sadržaj vezan za određenu temu. U tom trenutku sadržaj dobijaju samo oni korisnici koji su pretplaćeni na njega. U slučaju korišćenja *WAMP*-a, za implementaciju klijentske strane je potrebno korišćenje dodatne *JavaScript* biblioteke.

Instalacija Ratchet-a se najlakše vrši putem alata *PHP Composer*. Ovaj alat je zadužen za preuzimanje i nadogradnju svih komponenti *PHP* aplikacije. Zahvaljujući njemu moguće je vršiti instalaciju jednom komandom u terminalu operativnog sistema, a još jednom linijom u *PHP* kodu učitavanje svih potrebnih biblioteka.

7.3 Implementacija slučajeva korišćenja mobilne aplikacije

U ovom poglavlju je obrađena implementacija reprezentativnih slučajeva korišćenja klijentske mobilne aplikacije. Reprezentativni slučajevi korišćenja su oni slučajevi korišćenja koji demonstriraju korišćenje tehnologija obrađenih u radu. Svaki slučaj potrebe je u prethodnom poglavlju opisan osnovnim i alternativnim tokom. U nastavku slijede detalji implementacije konkretnog slučaja uz odgovarajuće slike ekrana i pregled poruka koje se razmijenjuju između klijenta i servera prilikom komunikacije.

7.3.1 Slučaj korišćenja – Registracija korisnika

Forma za registraciju je data na slici 11.

Odabir tehnologije i opis komunikacije:

Za ovaj slučaj korišćenja adekvatna veza između klijenta i servera je uspostavljena korišćenjem veb servisa. *WebSocket*-e nije potrebno koristiti pošto je očigledno

potrebna jednosmjerna komunikacija i ne radi se o vremenski kritičnoj operaciji, te bi u ovom slučaju samo došlo do nepotrebne komplikacije. Potiskivana obavještenja nisu potrebna u ovom slučaju zato što klijent pokreće zahtjev za registraciju, a server mu gotovo trenutno odgovara (u slučaju dobre internet veze). Naravno, i ona bi se mogla primijeniti u ovom slučaju ali bi samo došlo do komplikovanja prilično jednostavnih operacija, pošto bi bio potreban aplikacioni server koji podržava XMPP (zbog slanja poruka od klijenta ka serveru).

Poruka koju klijent šalje serveru je u sebi sadrži polja koja je klijent unio. Poruka se šalje *HTTP POST* zahtjevom na lokaciju /nalog. U slučaju da nalog sa unijetim brojem telefona ne postoji u bazi i da su podaci validni, server vraće aplikaciji *HTTP* odgovor sa statusom 201 (*Created*) i *JSON* sledećeg oblika:

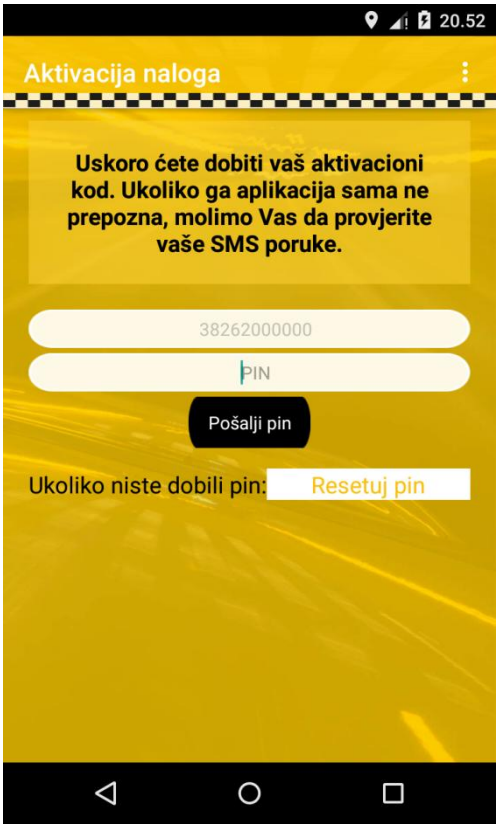
```
{
	korisnikIdNovi: 16969,
	token: "52449efc7aadff264831b9a5821216f109a90513"
}
```

Identifikator korisnika i token u aplikaciji služe za komunikaciju izmedju aplikacije i *WebSocket* servera. Ovaj token ne predstavlja token uređaja koji se dobija od strane *FCM* servera i služi za potiskivana obavještenja. Ukoliko već postoji nalog sa unijetim brojem telefona, server vraće *HTTP* kod 409 (*Conflict*). U slučaju neispravnih podataka server vraće kod 400 (*Bad request*). Nakon uspiješnog registrovanja korisnik se preusmjerava na ekran za aktivaciju naloga.

Slika 11 – Forma za registraciju

7.3.2 Slučaj korišćenja – Aktivacija naloga

Forma za aktivaciju je data na slici 12.



Slika 12 – Forma za aktivaciju naloga

Odabir tehnologije i opis komunikacije:

Za ovaj slučaj korišćenja adekvatna veza između klijenta i servera je uspostavljena korišćenjem veb servisa. *WebSocket*-e nije potrebno koristiti pošto je očigledno potrebna jednosmjerna komunikacija i ne radi se o vremenski kritičnoj operaciji, te bi u ovom slučaju samo došlo do nepotrebne komplikacije. Potiskivana obavještenja nisu potrebna u ovom slučaju zato što klijent ili sama aplikacija pokreću zahtjev za aktivaciju naloga, a server gotovo trenutno odgovara (u slučaju dobre internet veze).

Korisnik dobija kod za aktivaciju naloga SMS porukom. Aplikacija može sama da pročita sadržaj SMS poruke i iz nje izvuče aktivacioni kod. Potrebno je da aplikacija ima registrovan prijemnik javnog emitovanja (engl. *broadcast receiver*) za događaj **dolazna poruka**. Nakon prijema SMS poruke, operativni sistem šalje svim aplikacijama sa registrovanim prijemnikom obavještenje o novoj dolaznoj poruci, nakon čega aplikacije mogu da vide broj pošiljaoca, sadržaj poruke itd.

Poruka koju klijent šalje serveru prilikom aktivacije naloga u sebi sadrži token naloga koji treba aktivirati, vrijeme kada je primljen kod za aktivaciju poslednji put i sam kod za aktivaciju. Poruka se šalje *HTTP PUT* zahtjevom. U slučaju da je kod ispravan, server vraće aplikaciji *HTTP* odgovor sa statusom 200 (*OK*). U slučaju da je kod neispravan, server vraća kod 400 (*Bad request*).

Korisnik može da traži ponovno slanje aktivacionog koda. U tom slučaju se šalje *HTTP GET* zathjev koji sadrži ID naloga kome treba ponovo poslati kod. Ukoliko je kod uspješno poslat server šalje SMS sa kodom i vraća rezultat 200 (*OK*). Ukoliko je korisnik imao više od 10 zahtijeva za novim kodom u poslednja 24 časa ili je imao bar jedan u poslednjih 5 minuta server vraće kod 409 (*Conflict*) i korisnik se obaviještava koliko mora da sačeka prije nego zatraži novi kod.

7.3.3 Slučaj korišćenja – Prijava korisnika

Forma za prijavu korisnika je data na slici 13.

Odabir tehnologije i opis komunikacije:

Za ovaj slučaj korišćenja adekvatna veza između klijenta i servera je uspostavljena korišćenjem veb servisa. *WebSocket*-e nije potrebno koristiti pošto je očigledno potrebna jednosmjerna komunikacija i ne radi se o vremenski kritičnoj operaciji, te bi u ovom slučaju samo došlo do nepotrebne komplikacije. Potiskivana obavještenja nisu potrebna u ovom slučaju zato što klijent šalje zahtijev za prijavu na sistem, a server gotovo trenutno odgovara (u slučaju dobre internet veze).

Slika 13 – Forma za prijavu

Poruka koju klijent šalje ka serveru prilikom prijave na sistem u sebi sadrži broj telefona koji je vezan za nalog i lozinku. Poruka se šalje *HTTP GET* zahtjevom ka lokaciji /nalog. Ukoliko su podaci ispravni, dobija se *HTTP* odgovor čiji je status 200 (*OK*) i sledeći *JSON*:


```
{
    id: 5,
    brTel: "38269295646",
    ime: "Petar Rutesic",
    pass: "25d55ad283aa400af464c76d713c07ad",
    pol: 1,
    email: null,
    datumRodj: "1990-07-12",
    datumReg: "1467497298",
    datumAkt: "1476995740",
    status: 1,
    token: "38dae70e06f7c25d7a1829d42358c8d9a91cb457",
    resetToken: "oXNymfoL6B",
    deviceToken: "cdaTon4KldA:APA91bEApicwJZhDAv38hAFEY3_HMz3IJpGiy0m2YkaQE4fhoBEljkhcgY9ual-x831v2jfoOlk56eu4ZU9YPBMJfzBSXV5RewMslIB84gtWbw4ci2TF42uh4_-5TAdriPKB4nKGx5fc"
}
```


Ukoliko nalog ne postoji na serveru dobija se *HTTP* kod 404 (*Resource not found*) sa praznim tijelom odgovora. U slučaju nepravilnog zahtijeva vraća se kod 400 (*Bad Request*).

Na osnovu vrijednosti polja “status” se može očitati trenutno stanje naloga:

1. status=0 – nalog nije aktiviran i korisnik se preusmijerava na formu za aktivaciju
2. status=1 – nalog je aktiviran i korisnik se preusmijerava na glavnu formu za naručivanje taksi vozila
3. status=2 ili status=3 – korisničkom nalogu je zabranjen pristup aplikaciji

7.3.4 Slučaj korišćenja – Resetovanje lozinke

Forma za prijavu korisnika je data na slici 14.

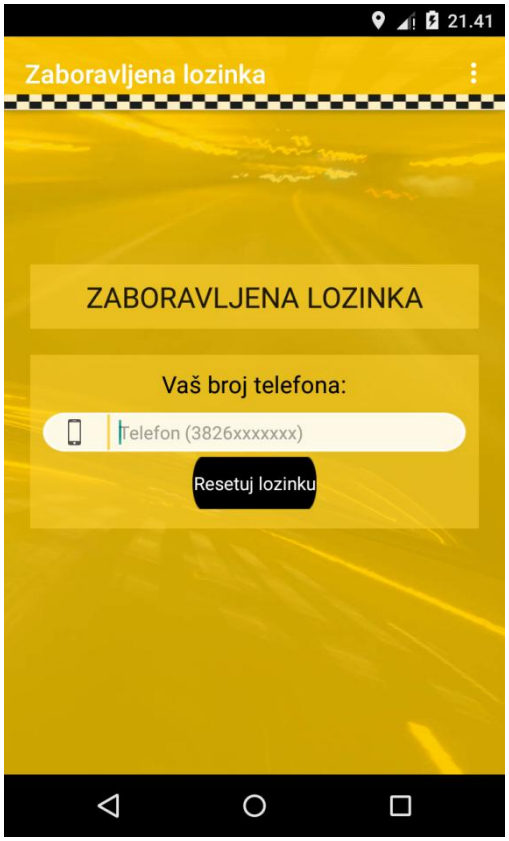
Odabir tehnologije i opis komunikacije:

Za ovaj slučaj korišćenja adekvatna veza između klijenta i servera je uspostavljena korišćenjem veb servisa. *WebSocket*-e nije potrebno koristiti pošto je očigledno potrebna jednosmjerna komunikacija i ne radi se o vremenski kritičnoj operaciji, te bi u ovom slučaju samo došlo do nepotrebne komplikacije. Potiskivana obavještenja nisu potrebna u ovom slučaju zato što klijent šalje zahtijev za dobijanje nove lozinke, a server gotovo trenutno odgovara (u slučaju dobre internet veze).

Zahtjev za resetovanje lozinke se šalje kao poruka koja u sebi sadrži broj telefona naloga čiju lozinku treba resetovati. Poruka se šalje kao *HTTP GET* zahtjev. Ukoliko je broj telefona vezan za neki nalog na serveru, dobija se *HTTP* odgovor čiji je status 200 (*OK*) i sledeći *JSON*:

```
{
    uspjesno: 1
}
```

U slučaju da je vrijednost uspjesno jednaka 1 to znači da je uspješno poslat e-mail sa *URL*-om za resetovanje lozinke; ukoliko je vrijednost jednaka 2 to znači da je poslat SMS umjesto e-maila. Ukoliko ne postoji nalog koji je vezan za unijeti broj telefona, kao dogovor se dobija *HTTP* kod 404 (*Resource not found*).



Slika 14 – Forma za resetovanje lozinke

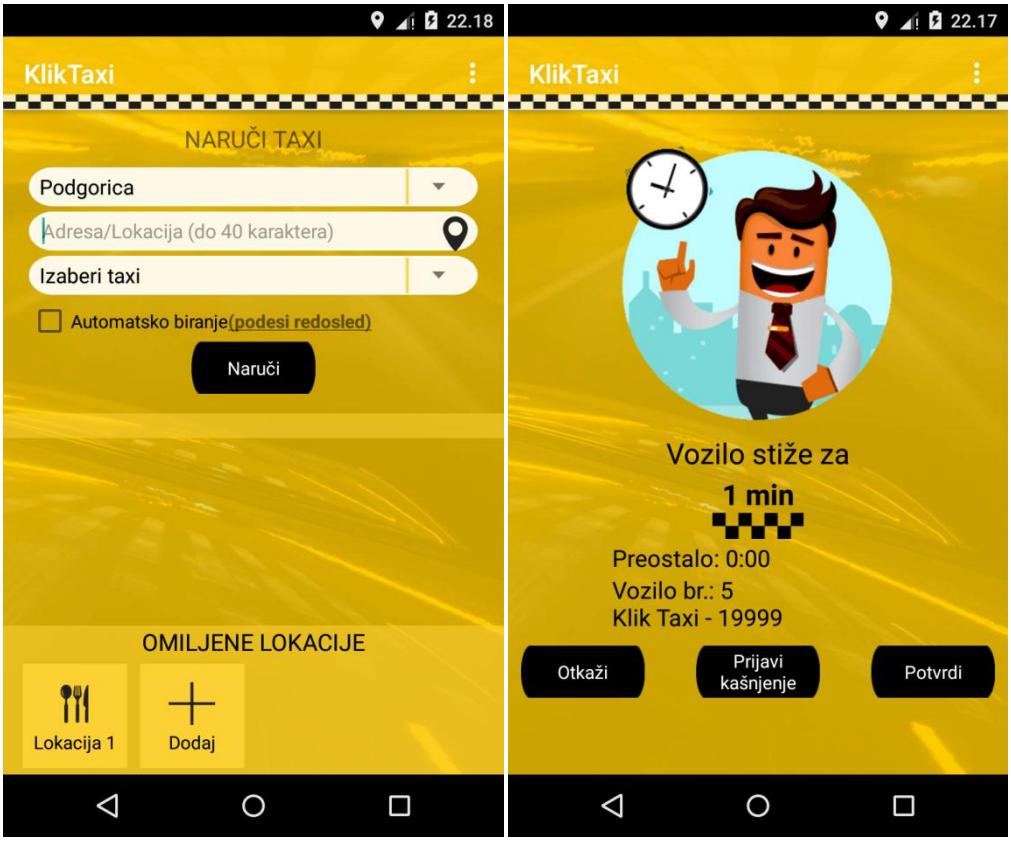
7.3.5 Slučaj korišćenja – Naručivanje vozila

Prikaz ekrana prilikom naručivanja vozila je dat na slici 15a. Slika 15b prikazuje izgled ekrana kada je korisniku dodijeljena vožnja.

Odabir tehnologije i opis komunikacije:

Ovaj slučaj korišćenja je zapravo srž KlikTaxi aplikacije. Jasno je da bi upotreba veb servisa ili potiskivanih obavještenja bila prilično komplikovana u ovom slučaju. Ovdje se ne radi o prostoj komunikaciji između klijenta i servera gdje klijent šalje zahtijev a

server odgovara na njega. Ovdje je server posrednik između dva klijenta (korisnika mobilne aplikacije i dispečera koji koristi veb aplikaciju).



Slika 15a - Forma za naručivanje vozila i Slika 15b – dodijeljeno vozilo

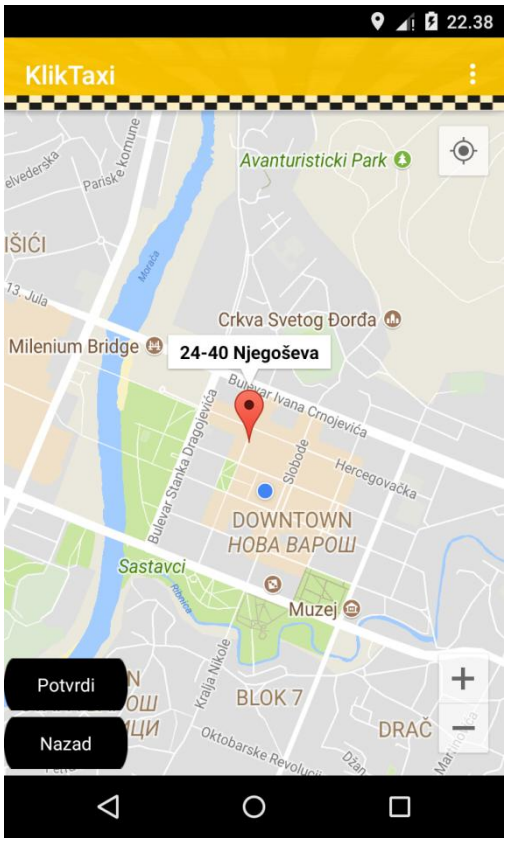
Odgovor dispečera se ne dešava odmah nakon zahtijeva korisnika, već može proteći i do par minuta prije nego dispečer odgovori. U ovom slučaju bi se veb servisom mogli slati zahtijevi ka serveru, pa potiskivanim obavještenjem proslijeđivati ka dispečerima. Ovakvo riješenje nije efikasno jer se koriste dvije tehnologije. Takođe, ni sa stanovišta potrošnje resursa nije optimalno - često će se raditi i povlačenje podataka na klijentima da bi se pratilo stanje što dodatno troši internet protok i bateriju uređaja.

Moguće je koristiti samo veb servise. U slučaju da se koriste samo veb servisi, potrebno bi bilo konstanto povlačiti podatke preko veb servisa i na dispečerskoj i na korisničkoj aplikaciji. I u tom slučaju, bespotrebno bi se trošio internet protok i baterija. Drugi, još veći problem, je određivanje intervala koji protekne između dva povlačenja podataka. Razlog tome je što za kratke intervale se može simulirati komunikacija u realnom vremenu uz veliko trošenje resursa, dok kod dugačkih intervala novi podaci ne bi bili odmah dostupni. Uzrok tome je što može proteći dosta vremena od momenta kada je neki podatak izmijenjen do momenta kada je zahtijevano povlačenje podataka.

Zbog svega navedenog kao bolje riješenje se logično nameću soketi, konkretno u ovom slučaju *WebSocket*. Zahvaljujući njima omogućena je dvosmijerna komunikacija u realnom vremenu između dispečera i korisnika, tako da se izmjene koje nastaju prilikom određenih događaja (odgovora na narudžbu, prijave kašnjenja

itd.) vrlo lako prate na korisničkim interfejsima. Problem ostaje samo prilikom prekida veze ka serveru, pa je pri sledećem uspostavljanju potrebno pokupiti trenutno stanje (aktivne vožnje, naručene – zaostale i sl).

Zahvaljujući tome što je *WebSocket* konekcija trajna, nije potrebno konstantno povlačiti podatke sa servera da bi se detektovali događaji (odobrena ili odbijena vožnja itd.) već njih server jednostavno prosljeđuje i klijentima i dispečerima.



Slika 16 – Odabir adrese korišćenjem Gugl mape

Kada korisnik uspostavi vezu sa *WebSocket* serverom, pokreće se metod *onOpen()* u mobilnoj aplikaciji. U tom metodu je definisano slanje *JSON* poruke ka soket serveru, koja je sledeće sadržine:

```
{
      kod: 301;
      id: "38dae70e06f7c25d7a1829d42358c8d9a91cb457"
}
```

Kod 301 označava da je u pitanju prijava korisnika na soket server, id označava identifikacioni token klijenta. Umjesto cjelobrojnog id-ja je stavljen token da bi se otežala lažna autentifikacija (npr. neko može nasumično da unosi cijelobrojne identifikatore i tako slučajno pogodi nečiji i lažno se prijavi na server). Ukoliko dati token nije vezan ni za jedan nalog u bazi podataka, aplikacija dobija nazad *JSON* poruku sa poljem kod: 304 i raskida konekciju ka soket serveru. Ukoliko su podaci ispravni, aplikacija dobija dvije *JSON* poruke sa kodovima 303 i 320. Poruka sa

kodom 303 nosi u sebi podatke o broju aktivnih vožnji korisnika (sve vožnje kojima nije definisan konačan status, što znači da nisu ni odbijene, ni otkazane, ni završene). Ukoliko takvih vožnji ima, korisniku se prikazuje ekran sa detaljima vožnje, dat na slici 15b. Poruka sa kodom 320 sadrži podatke o svim aktivnim taksi udruženjima i gradovima u sistemu, i na osnovu nje se popunjavaju polja sa spiskom gradova i taksi udruženja na glavnom ekranu. Takva poruka može imati sledeći oblik:

```
{
      "kod": 320 ,
      "taxiji": [
                  {
                  "id": 1 ,
                  "naziv": "Klik Taxi" ,
                  "gradId": 22 ,
                  "brTel": 19778 ,
                  "tarifa1": 0.45 ,
                  "tarifa2": 0.55 ,
                  "start": 0.0 ,
                  "cekanje": 0.1 ,
                  "minimal": 1.0
                  }
      ],
      "gradovi": [
                  {
                  "id": 1 ,
                  "naziv": "Podgorica"
                  }
      ]
}
```

Ukoliko je korisnik unio sve podatke, klikom na dugme “Naruči” može da pošalje zahtjev za vozilo ka dispečerima. U ovom slučaju aplikacija formira *JSON* poruku sa kodom 310 i sadržajem: adresom, identifikatorom taksi udruženja i gps koordinatama ako ih je moguće pronaći. Primjer takve poruke je sledeći:

```
{
      "kod": 310 ,
      "taxiId": 1 ,
      "gradId": 1 ,
      "lokacija": "Marka Markovica 11" ,
      "lat": 19.44732 ,
      "lon": 44.22334
}
```

Nakon prijema ovog zahtjeva server vrši unos podataka iz *JSON* poruke u bazu podataka i korisniku vraća *JSON* poruku sa identifikatorom vožnje, nakon čega prosleđuje zahtijev dispečerima. Primjer poruke:

```
{
	"kod": 216 ,
	"noviId": 1564462
}
```

Kada korisnik primi ovu poruku, prikazuje se štoperica koja odbrojava koliko je sekundi proteklo da korisnik čeka na odgovor dispečera (tj. čeka na prihvatanje ili odbijanje narudžbe). Primjer je dat na slici 17.

Slika 17 - Dispečer je dobio narudžbu

Dispečeri mogu da prihvate narudžbu i dodijele vozilo korisniku slanjem *JSON* poruke ka serveru. Server unosi podatke iz poruke u bazu podataka (mijenja status narudžbe, unosi broj vozila i broj minuta) i dalje prosljeđuje poruku sa sledećim sadržajem:

```
{
	"kod": 310 ,
	"taxiId": 1 ,
	"gradId": 1 ,
	"lokacija": "Marka Markovica 11" ,
	"lat": 19.44732 ,
	"lon": 44.22334 ,
	"lat": 19.44732 ,
	"id": 1564462,
	"brVozila": 112,
	"brMinuta": 1,
```

```
    "pracenje": 1
}
```

U prethodnoj poruci polje id označava identifikator narudžbe iz baze podataka a polje pracenje govori da li je moguće pratiti kretanje taksi vozila dok dolazi do odabrane adrese.

Dispečer može da odbije vožnju i o tome obavijesti korisnika slanjem *JSON* poruke ka serveru, koja sadrži neki od sledećih kodova:

- voznja odbijena - kod: 111
- vožnja odbijena zbog nejasne adrese - kod:114

Ukoliko je aplikacija aktivna u pozadini, korisnik dobija potiskivano obavještenje.

Nakon odobravanja vožnje, korisnik može da:

- prijavi kašnjenje - slanjem *JSON* poruke sa kodom 312 i identifikatorom vožnje
- otkaže vožnju - slanjem *JSON* poruke sa kodom 311 i identifikatorom vožnje
- potvrdi vožnju - slanjem *JSON* poruke sa kodom 320 i identifikatorom vožnje

Nakon slanja bilo koje od ovih poruka ka serveru, server vrši izmijene nad tabelom narudžbi i mijenja njihov status u: otkazane, u kašnjenju ili završene. Nakon izmijena, o tome obavještava korisnika slanjem *JSON* poruke sa odgovarajućim kodom:

- kod 218 – vožnja otkazana, dispečer obaviješten
- kod 219 – vožnja se više ne može otkazati
- kod 220 – vozilo kasni, dispečer je obaviješten
- kod 221 – vozilo još ne kasni, dispečer nije obaviješten

U svakom trenutku dok je korisnik povezan na soket, može da primi poruku sa nekim od sledećih kodova:

- kod 222 – korisniku je zabranjen pristup zbog psovanja
- kod 223 – korisnik je prijavljen jer se nije pojavio na adresi
- kod 224 – korisniku je zabranjen pristup jer se 3 puta nije pojavio na adresi u poslednjih 7 dana
- kod 101 – novi aktivni dispečer. Udruženje za koje dispečer radi se dodaje u spisak aktivnih udruženja ukoliko ranije nije bilo na spisku. U ovoj poruci osim koda se nalazi i taxiId, identifikator udruženja za koje dispečer radi
- kod 102 – udruženje je ostalo bez aktivnih dispečera i uklanja se sa spiska aktivnih udruženja. Poruka takođe sadrži taxiId
- kod 112 – poruka proizvoljnog sadržaja koju dispečer šalje korisniku. Sadrži polje sa tekstom koji dispečer unosi. Najčešće se dobija kao odgovor dispečera kada korisnik prijavi kašnjenje

7.3.6 Slučaj korišćenja – Upravljanje nalogom

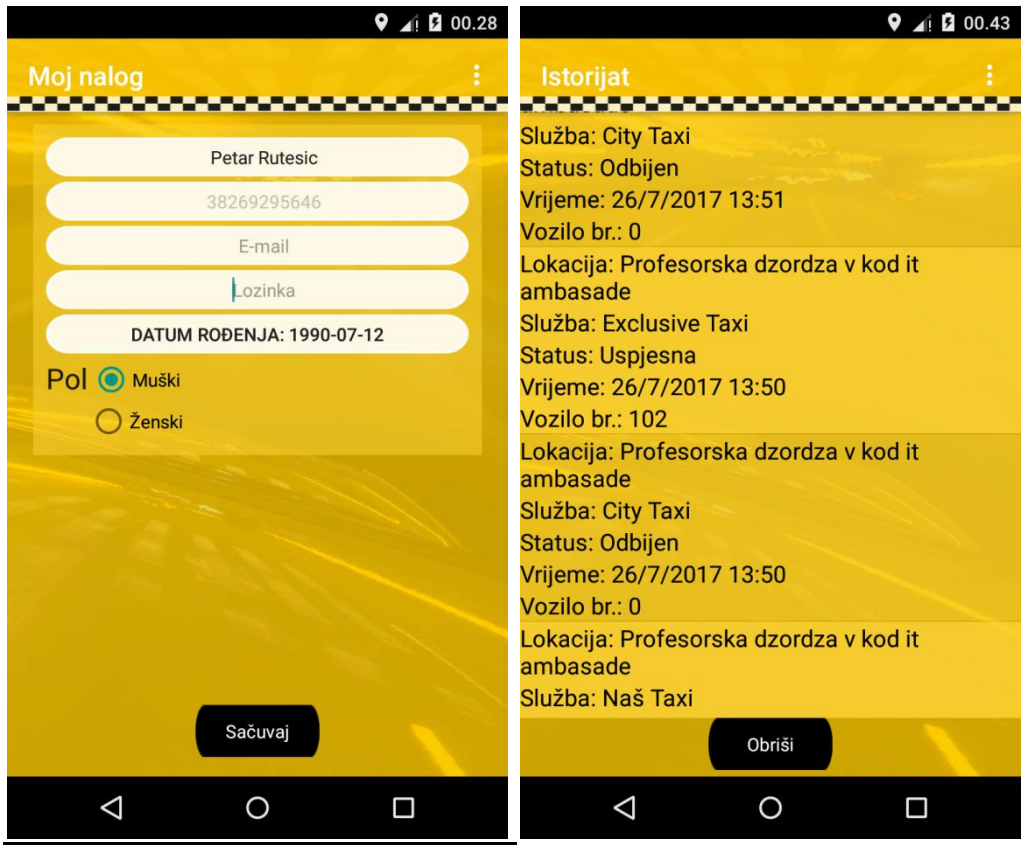
Primjer forme za izmijenu podataka je dat na slici 18.

Odabir tehnologije i opis komunikacije:

Za ovaj slučaj korišćenja adekvatna veza između klijenta i servera je uspostavljena korišćenjem veb servisa. *WebSocket*-e nije potrebno koristiti pošto je očigledno potrebna jednosmjerna komunikacija i ne radi se o vremenski kritičnoj operaciji, te bi u ovom slučaju samo došlo do nepotrebne komplikacije. Potiskivana obavještenja nisu potrebna u ovom slučaju zato što klijent šalje zahtijev za izmijenu podataka, a server gotovo trenutno odgovara (u slučaju dobre internet veze).

Zahtjev za izmijenu podataka naloga se šalje kao poruka koja u sebi sadrži sve podatke sa forme za izmijenu naloga. Poruka se šalje kao *HTTP PUT* zahtjev. Ukoliko su svi podaci ispravni i unos u bazu podataka je uspješno obavljen, dobija se *HTTP* odgovor čiji je status 200 (*OK*) i prazno tijelo odgovora. U slučaju da izmijene nisu sačuvane ili su podaci nevalidni, dobija se *HTTP* odgovor čiji je status 400 (*Bad request*) i ispisuje se odgovarajuća poruka na ekranu.

7.3.7 Slučaj korišćenja – Pregled narudžbi



Slika 18 – Forma za izmjenu podataka i Slika 19 – Pregled narudžbi

Primjer forme za pregled narudžbi je dat na slici 19.

Odabir tehnologije i opis komunikacije:

Za ovaj slučaj korišćenja adekvatna veza između klijenta i servera je uspostavljena korišćenjem veb servisa. *WebSocket*-e nije potrebno koristiti pošto je očigledno potrebna jednosmjerna komunikacija i ne radi se o vremenski kritičnoj operaciji, te bi u ovom slučaju samo došlo do nepotrebne komplikacije. Potiskivana obavještenja nisu potrebna u ovom slučaju zato što klijent šalje zahtijev za pregled podataka, a server gotovo trenutno odgovara (u slučaju dobre internet veze).

Zahtjev za pregled narudžbi se šalje kao poruka koja u sebi sadrži identifikator korisnika korisnika. Poruka se šalje kao *HTTP GET* zahtijev. Ukoliko je zahtijev ispravan, dobija se *HTTP* odgovor čiji je status 200 (*OK*) i *JSON* poruka sa nizom narudžbi.

7.3.8 Slučaj korišćenja – Upravljanje obavještenjima

Primjer pregleda obavještenja je dat na slici 20.

Odabir tehnologije i opis komunikacije:

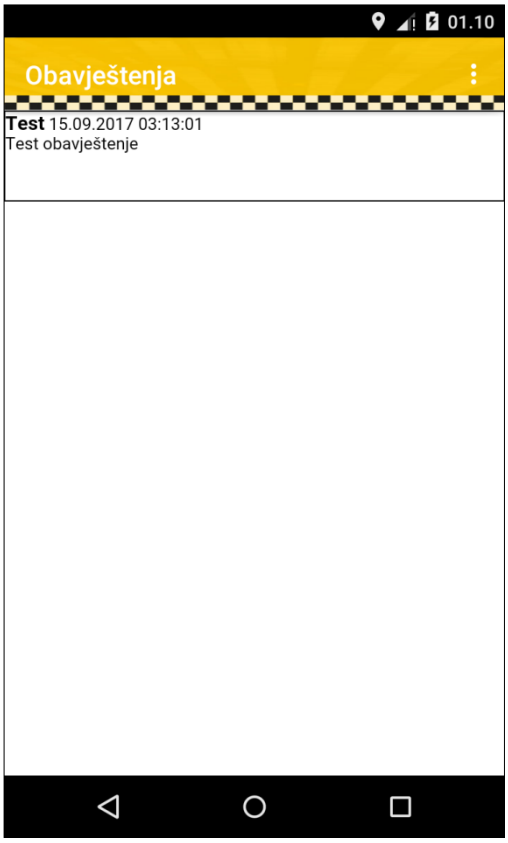
Za ovaj slučaj korišćenja korišćena su potiskivana obavještenja i povlačenje podataka. Kada administrator sistema unese novo obavještenje u bazu podataka, cilj je da ono stigne korisniku bez obzira da li mu je aplikacija upaljena ili nije. Jedini način da se ovo postigne jeste potiskivanim obavještenjima. Takvo obavještenje se šalje tako što aplikacioni server nakon unosa obavještenja u bazu šalje sledeći *HTTP POST* zahtijev ka *FCM* serveru:

```
https://fcm.googleapis.com/fcm/send
Content-Type:application/json
Authorization:key=AIzaSyZ-1u...0GBYzPu7Udno5aA

{
  "registration_ids" : ["APa43bHu2n4M34egoK2t2KZ34FBaFUH-1RYqx...",
"APa43bHu2n4M34egoK2t2KZ34FBaFUH-1RYqx..."],
  "data" : {
    "naslov" : "Naslov",
    "poruka" : "Sadržaj obavještenja"
  }
}
```

U primjeru zahtjeva nema polja notification, zato što se u KlikTaxi aplikaciji notifikacija pokazuje i kad je aplikacija aktivna na ekranu (o ovome je bilo riječi u poglavlju 4.5.2). Zapravo, KlikTaxi aplikacija dobija samo poruku od *FCM* servisa i onda sama pravi lokalno potiskivano obavještenje.

Pregled obavještenja se obavlja klikom na potiskivano obavještenje ili tako što korisnik odabere stavku “Obavještenja” u meniju aplikacije. Nakon toga, šalje se *HTTP GET* zahtijev ka serveru za preuzimanje stranice obavještenja.php. Kao



Slika 20 – Pregled obavještenja

odgovor od servera se dobija *HTML* strana koja se učitava u *WebView* komponentu aplikacije. *WebView* je jedna od osnovnih komponenti Android platforme, koja omogućava pregledač veba unutar aplikacije na jednostavan način, dovoljno je samo zadati *URL* veb stranice.

8 Zaključak

U prethodnim poglavljima obrađene su različite tehnike povezivanja mobilnih aplikacija i servera: veb servisi, soketi i potiskivana obavještenja. Opisane su osnovne karakteristike različitih vrsta veb servisa, sa posebnim akcentom na *REST* servise. Na osnovu izrečenog, može se zaključiti da su veb servisi osnova platformski nezavisnih aplikacija. Istaknute su osobine *REST*-olikih servisa, zahvaljujući kojima je *REST*, uz adekvatno poznavanje osnovnih principa rada veb servisa, prilično jednostavan za implementaciju.

Kao alternativa veb servisima, javlja se povezivanje aplikacije i servera putem *WebSocket*-a. Oni svoje korijene imaju u klasičnim mrežnim soketima. Opisana je i sama specifikacija protokola i istaknuta jednostavnost implementacije, naročito kada su u pitanju moderni pregledači veba. Uz određene alternative, *WebSocket*-i predstavljaju dobro riješenje u slučaju kada je potrebna dvosmijerna komunikacija između servera i klijenta u realnom vremenu.

Kao treće riješenje, opisana su potiskivana obavještenja. Svaka mobilna platforma ima sopstveni servis za slanje obavještenja. Posebno se izdvaja Guglov *FCM* servis zato što istovremeno podržava rad na više platformi.

Na kraju je dat pregled arhitekture i implementacije mobilne aplikacije KlikTaxi. KlikTaxi aplikacija se sastoji od tri aplikacije: klijentske mobilne aplikacije, dispečerske veb aplikacije i administratorske veb aplikacije. Njena namjena je da podrži i olakša rad dispečerima taksi udruženja, a pritom i da obezbijedi korisnicima besplatan i efikasan način za pozivanje taksi vozila.

Razvojno okruženje za Android omogućava jednostavnu integraciju sa servisom za potiskivana obavještenja *FCM*, a uz odgovarajuće biblioteke je lako implementirati veb servise i *WebSocket*.

Kroz primjere implementacije su prikazani reprezentativni slučajevi primjene neke od obrađenih tehnika, uz diskusiju o izboru tehnike u zavisnosti od samog slučaja korišćenja. Na osnovu tih primjera, može se zaključiti da su veb servisi jednostavno i dobro riješenje kada klijent šalje neki zahtijev ka serveru, a server mu odmah odgovara na zahtijev. U slučaju kada je potrebna komunikacija u realnom vremenu između klijentske i dispečerske aplikacije, efikasnije je koristiti *WebSocket*-e. Oni omogućavaju štednju resursa: internet protok i baterija se ne troši bespotrebno pošto nema uspostavljanja i raskidanja konekcije pri svakom zahtijevu. Kada nije sigurno da li je aplikacija na uređaju aktivna ili nije, a potrebno je da korisnik odmah dobije neko obavještenje, najbolje riješenje jesu potiskivana obavještenja. Implementacija istih je jednostavna u Android aplikacijama. Podržavaju ih sve moderne platforme za pametne telefone. Kao i *WebSocket*-i, i ona čuvaju resurse uređaja.

9 Reference

[1] W3C Working Group, "Web Services Architecture", 2004.

[2] L. Richardson / S. Ruby, *REST*ful Web Services, 1 yp., O'Reilly, 2007.

[3] M. Kalin, Java Web Services: Up and Running, 2nd ed.

[4] R. Fielding, "*REST APIs* must be hypertext driven". Dostupno na: *http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven.* [01 07 2017].

[5] I. Grigorik, "High Performance Browser Networking". Dostupno na: *https://hpbn*.co/. [25 6 2017].

[6] R. Fielding, "Architectural Styles and the Design of Network-based Software Architectures", Doktorska disertacija, 2000.

[7] *JSON*.org, "*JSON* org". Dostupno na: *http://www.json.org/*. [30 6 2017].

[8] W3C, "*XML* 1.0 specification". Dostupno na: *https://www.w3.org/TR/2008/REC-xml-20081126/*. [30 6 2017].

[9] Network Working Group, "Uniform resource identifier *URI* syntax - RFC3986" 2005. Dostupno na: *https://tools.ietf.org/html/rfc3986*. [6 2017].

[10] V. Pterneas, Getting Started with *HTML*5 WebSocket Programming, Birmingham: Packt Publishing, 2013.

[11] IETF, "Generic Event Delivery Using *HTTP* Push".

[12] IETF, "WebSocket Protocol RFC6455".

[13] Apple inc., "APNs official documentation" Apple. Dostupno na: *https://developer.apple.com/go/?id=push-notifications*. [26 08 2017].

[14] Google inc., "*FCM* official documentation" Firebase. Dostupno na: *https://firebase.google.com/docs/cloud-messaging/concept-options*. [01 09 2017].

[15] D. Alex, R. Holly, J. Hildenbrand and A. Martonik, "Android History". Dostupno na: *https://www.androidcentral.com/android-history.* [9 2017].

[16] P. Ryan, "Dream(sheep++): A developer's introduction to Google Android" 23 9 2009. Dostupno na: *http://arstechnica.com/open-source/reviews/2009/02/an-*

introduction-to-google-android-for-developers.ars. [2017].

[17] R. Grasser and V. Matos, "Building Applications for the Android OS Mobile Platform".

[18] Ratchet*PHP*, "Ratchet*PHP*". Dostupno na: *http://socketo.me/docs/flow*.