

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 4090

**Aplikacija za dojavu događaja na
uređajima s operacijskim sustavom
Android**

Borna Sirovica

Zagreb, lipanj 2015

Zagreb, 11. ožujka 2015.

ZAVRŠNI ZADATAK br. 4090

Pristupnik: **Borna Sirovica (0036472287)**
Studij: Računarstvo
Modul: Računalno inženjerstvo

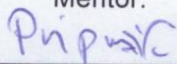
Zadatak: **Aplikacija za dojavu događaja na uređajima s operacijskim sustavom Android**

Opis zadatka:

Vaš zadatak je proučiti i analizirati programsku podršku za protokol MQTT u operacijskom sustavu Android. Ovaj protokol omogućava raspodijeljenu razmjenu poruka između krajnjih čvorova na principu objavi-pretplati. Prilikom objave poruke na određenom kanalu, ona će se isporučiti svim pretplatnicima tog kanala. Usporedbu objava i pretplata vrši posrednik MQTT. Proučite postojeće posrednike MQTT otvorenog koda izvedene u programskom jeziku Java te odaberite jedan koji ćete koristiti u vašoj aplikaciji. Korištenjem analizirane programske podrške oblikujte, programski izvedite i testirajte aplikaciju za dojavu događaja na uređajima s operacijskim sustavom Android. Svü potrebnu literaturu i uvjete za rad osigurat će Vam Zavod za telekomunikacije.

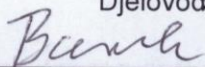
Zadatak uručen pristupniku: 13. ožujka 2015.
Rok za predaju rada: 12. lipnja 2015.

Mentor:



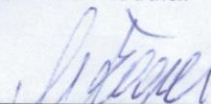
Doc. dr. sc. Krešimir Pripužić

Djelovođa:



Prof. dr. sc. Danko Basch

Predsjednik odbora za
završni rad modula:



Prof. dr. sc. Mario Žagar

Sadržaj

Uvod.....	1
1. Protokol MQTT.....	2
1.1. Model objavi-pretplati.....	3
1.2. MQTT konekcija.....	4
1.3. MQTT pretplata i objava.....	6
1.4. MQTT tema.....	11
2. Programska podrška za protokol MQTT.....	13
3. Posrednici MQTT otvorenog koda.....	14
4. O razvijenoj aplikaciji.....	15
4.1. Dizajn sustava.....	15
4.2. Tehničke značajke.....	17
4.3. Programske značajke.....	19
4.4. Funkcionalni zahtjevi.....	22
4.5. Upute za instalaciju.....	25
4.6. Konfiguriranje poslužitelja za razvijenu aplikaciju.....	26
Zaključak.....	30
Literatura.....	31
Sažetak.....	32

Uvod

Cilj ovog rada bio je proučiti, analizirati i implementirati programsku podršku za protokol MQTT u operacijskom sustavu Android te uz korištenje odabrane podrške oblikovati, programski izvesti i testirati aplikaciju za dojavu događaja na uređajima s operacijskim sustavom Android. Protokol MQTT je otvoren, jednostavan i lagan protokol koji omogućava raspodijeljenu razmjenu poruka između krajnjih čvorova na principu objavi-pretplati (*publish/subscribe*). Navedene karakteristike čine ga idealnim za korištenje u ograničenim okruženjima i mrežama niske propusnosti s ograničenim mogućnostima obrade, malim kapacitetom memorije i visokom latencijom. Objava poruka vrši se na određenom kanalu preko kojeg se poruka prosljeđuje svim pretplatnicima tog kanala. Za usporedbu objava i pretplata zadužen je posrednik MQTT (*MQTT broker*) koji prosljeđuje objavljene poruke i pamti pretplate klijenata. Detaljan opis protokola MQTT verzije 3.1 implementiranog u Android aplikaciji dan je u prvom poglavlju ovog završnog rada, dok su sljedeća dva poglavlja koncentrirana na analizu posrednika MQTT otvorenog koda i postojeće MQTT programske podrške pisane u programskom jeziku Java. U četvrtom poglavlju pobliže je opisana razvijena aplikacija te njena funkcionalnost i tehničke značajke.

1. Protokol MQTT

Protokol MQTT (*Message Queue Telemetry Transport*) razvili su Andy Stanford-Clark (IBM) i Arlen Nipper (Eurotech) 1999. godine u svrhu praćenja naftovoda usred pustinje [1]. Cilj im je bio razviti protokol koji bi omogućio efikasnu brzinu prijenosa podataka uz malu potrošnju baterije i manju cijenu cjelokupnog sustava.

Razvijeni protokol MQTT s objavi-pretplati arhitekturom pokazao se lakšim i ekonomičnijim od protokola HTTP (*HyperText Transfer Protocol*) s modelom zahtjeva i odgovora. Model objavi-pretplati pogonjen je događajima i omogućava klijentima da objavljuju poruke bez brige o njihovoj konačnoj destinaciji. Konačnu destinaciju objavljene poruke određuje posrednik MQTT koji na osnovu klijentskih pretplata prosljeđuje poruke i tako oslobađa klijente od slanja zahtjeva za porukama od interesa, za razliku od protokola HTTP kod kojeg klijenti moraju tražiti informacije koje trebaju od poslužitelja. Protokol MQTT odlikuje i mala veličina fiksnog zaglavlja (*fixed header*) od svega 2 bajta što dodatno smanjuje mrežno opterećenje [2].

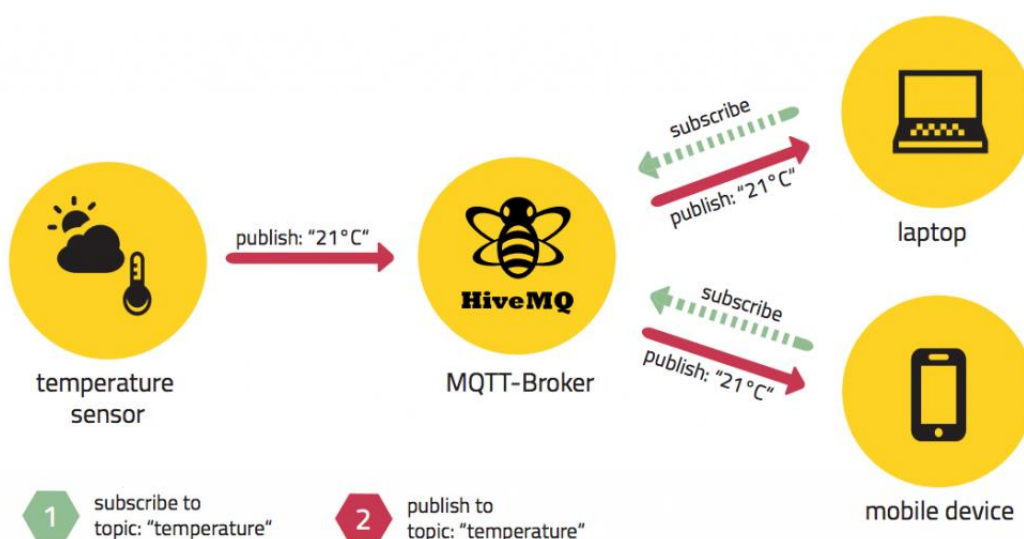
Prema članku Stepena Nicholasa protokol MQTT nudi uštedu baterije na mobilnim uređajima od oko 4.1% u jednom danu uz održavanje stabilne konekcije naspram protokola HTTP [3]. Iako inicijalno spajanje MQTT klijenta na poslužitelj troši više baterije jer se dodatno šalje i univerzalni identifikator klijenta, održavanje trajne konekcije uz primanje i objavljivanje poruka pokazalo se ekonomičnijim za protokol MQTT zbog toga što HTTP puno češće obavlja operaciju spajanja koja utječe na život baterije (slika 1).

Characteristics		3G		WiFi	
		HTTPS	MQTT	HTTPS	MQTT
Receive Messages	Messages / Hour	1,708	160,278	3,628	263,314
	Percent Battery / Hour	18.43%	16.13%	3.45%	4.23%
	Percent Battery / Message	0.01709	0.00010	0.00095	0.00002
	Messages Received (Note the losses)	240 / 1024	1024 / 1024	524 / 1024	1024 / 1024
Send Messages	Messages / Hour	1,926	21,685	5,229	23,184
	Percent Battery / Hour	18.79%	17.80%	5.44%	3.66%
	Percent Battery / Message	0.00975	0.00082	0.00104	0.00016

Slika 1. usporedba karaktetristika protokola MQTT i HTTP pri slanju i primanju 1024 poruke veličine 1 bajt [3]

1.1. Model objavi-pretplati

Kao što je spomenuto u prethodnom odlomku komunikacija između klijenata kod protokola MQTT ostvarena je obrascem objavi-pretplati uz središnju ulogu posrednika MQTT koji prati pretplate klijenata i delegira poruke u ovisnosti o pretplatama. Kada klijent šalje (*publish*) poruku posredniku mora uz poruku navesti temu (*topic*) na koju želi objaviti poruku. Tema posredniku predstavlja glavnu informaciju za usmjeravanje poruke prema krajnjim klijentima. S druge strane kada se klijenti pretplaćuju (*subscribe*) moraju navesti temu na koju se žele pretplatiti kako bi posrednik znao isporučiti sve poruke čija se tema podudara s temama na koje je klijent pretplaćen (slika 2).



Slika 2. *jednostavan opis MQTT objavi-pretplati arhitekture* [2]

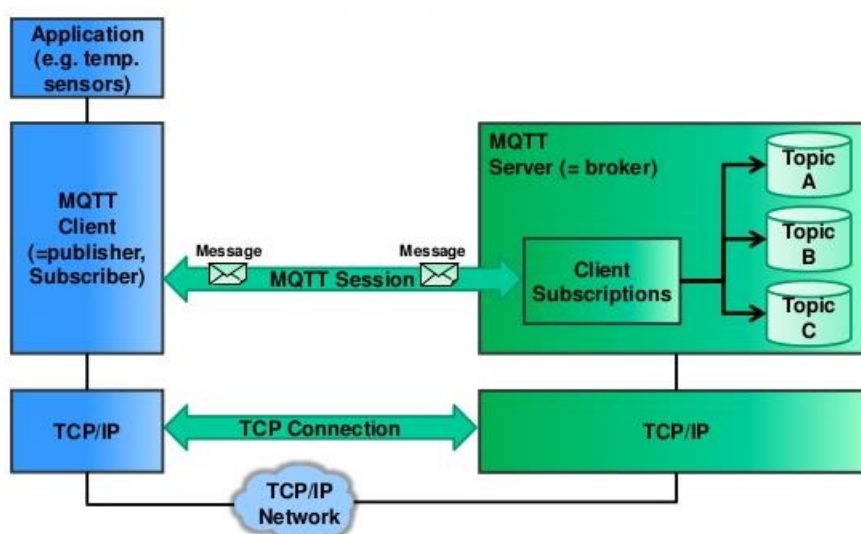
Kod ovakvog načina komunikacije klijenti se ne trebaju međusobno poznavati i mogu komunicirati samo preko odabranih tema čime se uklanja ovisnost između proizvođača (*producer*) i potrošača (*consumer*) informacija što povećava skalabilnost sustava tj. učinkovitost sustava bez obzira na značajno povećanje posluživanih resursa i/ili broja posluživanja.

Odvajanje pošiljatelja od primatelja može se podijeliti na tri dimenzije:

- Prostorna dimenzija – pošiljatelj i primatelj ne trebaju se međusobno poznavati (npr. prema IP adresi i portu)
- Vremenska dimenzija – pošiljatelj i primatelj ne trebaju biti aktivni u isto vrijeme
- Sinkronizacijska dimenzija – operacije pošiljatelja i primatelja nisu zaustavljene tijekom objave i primanja poruka (klijenti rade asinkrono neovisno jedan drugome)

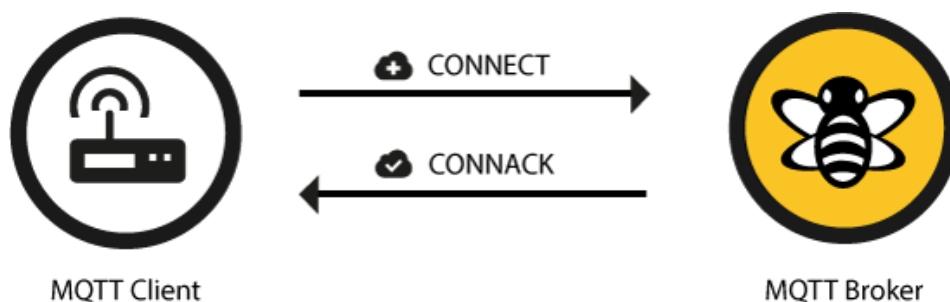
1.2. MQTT konekcija

Početak svake komunikacije među MQTT klijentima njihovo je spajanje na posrednika MQTT koji predstavlja središte komunikacije i nosi odgovornost za primanje svih poruka, njihovo filtriranje i prosljeđivanje svim pretplaćenim klijentima. Prilikom spajanja s posrednikom klijenti moraju navesti adresu posrednika i port 1883 koji je rezerviran isključivo za komunikaciju protokolom MQTT. Kako je MQTT baziran nad TCP/IP transportnim, odnosno mrežnim slojem (slika 3) svaki klijent prilikom spajanja na posrednika otvara trajnu TCP vezu iznad koje se otvara MQTT sesija za objavljivanje i primanje poruka.



Slika 3. *MQTT kao aplikacijski protokol iznad TCP transportnog protokola i IP mrežnog protokola* [2]

Samu konekciju inicira klijent koji šalje *Connect* poruku posredniku MQTT na koju on odgovara *Connack* porukom sa statusnim kodom koji daje informaciju o uspješnosti spajanja klijenta (slika 4) [5]. Nakon spajanja veza se održava trajno sve dok klijent ne odluči prekinuti vezu ili se konekcija ne izgubi.



Slika 4. *Uspostavljanje MQTT konekcije između klijenta i posrednika* [4]

Ako dođe do prekida konekcije na bilo koji način MQTT posrednik može pohraniti pristigle poruke i proslijediti ih klijentu kada se on ponovno spoji. Prilikom slanja *connect* poruke klijent mora definirati jedinstveni identifikator (*clientId*), pomoću kojeg posrednik razlikuje klijente. Uz jedinstveni identifikator klijent može postaviti i dodatne zastavice koje proširuju mogućnosti protokola MQTT. Zastavice koje se opcionalno mogu postaviti su [5]:

- Zastavica za čistu sesiju (*clean session flag*) – ako je ova zastavica postavljena MQTT posrednik će nakon prekida konekcije izbrisati sve informacije vezane uz prethodnu sesiju. U slučaju da zastavica nije postavljena uspostaviti će se trajna sesija i posrednik će pohraniti sve pretplate i propuštene poruke usprkos prekidu konekcije.
- Zastavice za korisničko ime i lozinku – dopuštaju korisniku autorizaciju postavljanjem korisničkog imena i lozinke.
- Zastavica oporuke (*will flag*) – ako je ova zastavica postavljena posrednik sprema poruku oporuke (*will message*) i šalje ju u drugim klijentima kada klijent s postavljenom zastavicom oporuke iznenadno izgubi konekciju s posrednikom.

Klijent uz ove zastavice može postaviti i dvobajtni period života konekcije (*keep alive*) koji predstavlja najveći dopušteni vremenski interval, između završetka slanja jednog i početka slanja drugog MQTT paketa, na kraju kojeg će konekcija biti prekinuta. Kako ne bi došlo do isteka ovog intervala odgovornost klijenta je da šalje jednostavne *Pingreq* pakete samo kako bi održao konekciju živom. Posrednik na *Pingreq* pakete odgovara *Pingresp* paketima kojima potvrđuje da normalno funkcioniра. Nakon što klijent MQTT posredniku pošalje *Connect* paket posrednik mora odgovoriti *Connack* paketom kako bi se konekcija uspješno uspostavila. *Connack* odgovor posrednika sadrži dvije informacije:

- Zastavicu koja označava postoji li sjednica (*session present flag*) –kada je zastavica postavljena znači da već postoji sesija između klijenta i posrednika tj. da su klijent i posrednik već imali zapamćenu interakciju prije trenutne konekcije. Kada zastavica nije postavljena stvara se nova prazna sesija. Ako pri konekciji klijent postavi zastavicu za čistu sesiju ova zastavica nikad neće biti postavljena za konekcije tog klijenta s posrednikom.
- Povratni kod – označava je li konekcija prihvaćena od strane posrednika, ako je tada je povratna vrijednost 0.

Nakon što se uspješno spoji na posrednika, klijent ima mogućnosti pretplate na nove teme i objave poruka. Navedene mogućnosti detaljnije će biti opisane sljedećim odlomkom.

1.3. MQTT pretplata i objava

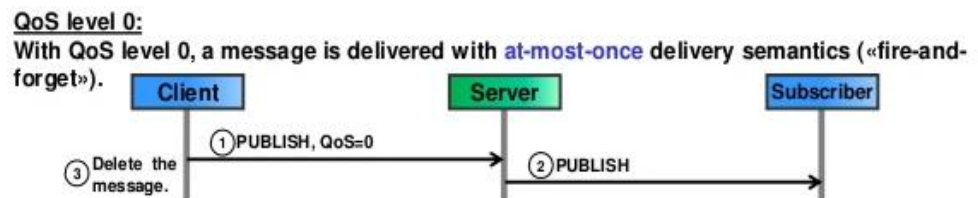
Kada klijent, koji se prethodno uspješno povezao s posrednikom, odluči objaviti poruku mora uz podatke koje želi objaviti navesti i temu na kojoj ju želi objaviti kako bi MQTT posrednik znao kojim klijentima proslijediti dobivenu poruku. Podaci unutar poruke (*payload*) šalju se posredniku u bajtovima jer MQTT pošiljateljima omogućava objavu podataka u bilo kojem obliku (binarni i tekstualni podaci, XML ili JSON) [5]. Objava poruke uključuje slanje paketa objave (*publish packet*) posredniku (slika 5).



Slika 5. *Paket objave* [6]

Taj paket uključuje sljedeće attribute:

1. Ime teme – jednostavan niz znakova strukturiran hijerarhijski s kosim crtama (*forward slash*) kao graničnicima (*delimiter*) koji posredniku predstavlja glavnu informaciju za daljnje usmjeravanje poruke.
2. QoS, razina kvalitete servisa (*Quality of Service Level*) – predstavlja razinu garancije isporuke objavljenih poruka. Ovaj dvobitni atribut može poprimiti 3 vrijednosti:
 - Vrijednost 0 (*at-most-once delivery*) – najniža razina kvalitete servisa kod koje se poruka isporučuje u skladu s mogućnostima protokola nižeg sloja (protokol TCP). Poruka je poslana samo jednom bez povratne informacije s primateljeve strane i bez spremanja i ponovnog slanja sa strane pošiljatelja (Slika 6).

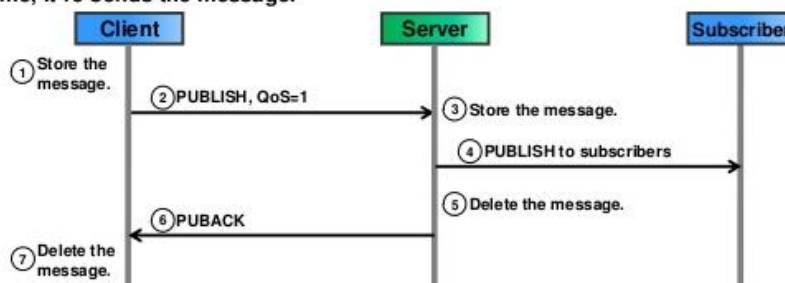


Slika 6. *Objava poruke uz QoS=0* [2]

- Vrijednost 1 (*at-least-once delivery*) – ova razina kvalitete servisa garantira da će poruka biti isporučena primateljima najmanje jednom, ali postoji mogućnost da bude isporučena i više puta. Pošiljalatelj prvo interno sprema poruku, a zatim ju šalje posredniku koji poruku prosljeđuje svim pretplatnicima i vraća potvrdu o slanju poruke u obliku *Puback* paketa koji sadrži identifikator paketa objavljene poruke (*packetId*). Nakon što primi potvrdu o uspješnoj objavi, pošiljalatelj iz internog spremnika briše poruku koju je spremio prije same objave (Slika 7).

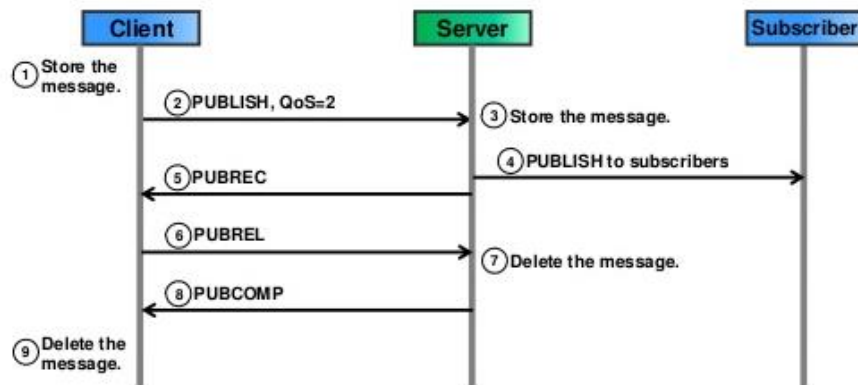
QoS level 1:

QoS level 1 affords *at-least-once* delivery semantics. If the client does not receive the PUBACK in time, it re-sends the message.



Slika 7. *Objava poruke uz QoS=1* [2]

- Vrijednost 2 (*exactly-once delivery*) – najviša, ali ujedno i najsporija razina kvalitete servisa koja garantira da će poruka biti isporučena točno jednom. Pošiljalatelj sprema poruku prije slanja posredniku koji, nakon što ju spremi i proslijedi primateljima, vraća potvrdu o objavi u obliku *Pubrec* paketa s identifikatorom paketa objavljene poruke. Nakon toga pošiljalatelj šalje *Pubrel* paket na koji posrednik odgovara *Pubcomb* paketom, oba paketa sadrže samo identifikatore objavljene poruke. Kako bi se osigurala objava poruke točno jedanput posrednik pamti identifikator poruke sve dok ne primi *Pubrel* paket nakon kojeg pošiljalatelj sigurno neće ponovno poslati poruku (Slika 8).



Slika 8. *Objava poruke uz QoS=2* [2]

3. Zastavica za pamćenje poruke (*retain flag*) – ova zastavica određuje hoće li posrednik zapamtiti poruku za određenu temu. Ako je zastavica postavljena, poruka će biti isporučena i klijentima koji se pretplate na temu poruke nakon njene objave. Treba nadodati da posrednik pamti samo jednu poruku za svaku temu pa će poruka biti spremljena na posredniku sve dok klijent na istu temu ne objavi novu poruku s postavljenom zastavicom za pamćenje.
4. Podaci (*payload*) – sadržaj poruke koja se šalje. Može biti u obliku slika, teksta s bilo kojim načinom kodiranja, šifriranih ili binarnih podataka.
5. Identifikator paketa (*packet id*) – dvobajtni podatak koji jedinstveno određuje poruku u komunikaciji klijenta i posrednika. Bitan je samo za pakete objave koji imaju QoS postavljen na vrijednost veću od 0.
6. Zastavica za prepoznavanje dupliciranih poruka (*DUP flag*) – Ako je ova zastavica postavljena znači da je poruka ponovno poslana jer pošiljalac nije primio potvrdu o uspješnoj objavi poruke od posrednika. Ova zastavica relevantna je samo za pakete objave koji imaju QoS veći od 0.

Nakon što paket objave s odgovarajućom porukom i temom dođe do posrednika, on na osnovu teme i zapamćenih pretplata proslijeđuje poruku. Kako bi znao kome proslijediti poruku posrednik mora omogućiti klijentima koji žele primati poruke mogućnost pretplate na teme koje ih zanimaju. Klijenti se pretplaćuju na temu slanjem paketa pretplate (*subscribe packet*) MQTT posredniku (slika 9).



Slika 9. *Paket pretplate* [6]

Paket pretplate sadrži identifikator paketa i listu pretplata. Identifikator paketa jedinstveno određuje paket u komunikaciji posrednika i pretplatnika, a lista pretplata sadrži listu tema na koje se klijent želi pretplatiti s razinama kvalitete servisa za svaku temu. Ovaj paket mora sadržavati najmanje jednu temu uz pripadajući QoS. Nakon što klijent pošalje paket pretplate s listom tema od interesa, posrednik mora vratiti odgovor u obliku *Suback* paketa koji potvrđuje uspješnu pretplatu (slika 10).



Slika 10. *Suback paket* [6]

Ovaj paket sadrži isti identifikator paketa kao i paket pretplate te listu povratnih kodova za svaku temu na koju se klijent želi prijaviti. Povratni kod može poprimiti 4 vrijednosti: 0, 1 i 2 za uspješno postavljene pretplate s razinom kvalitete servisa 0, 1 i 2 te vrijednost 128 ako pretplata nije uspjela [5].

Nakon uspješno postavljene pretplate klijent će od posrednika dobivati sve poruke čija tema se podudara s temom na koju se klijent pretplatio. Ako klijent ne želi više primati poruke s određene teme može otkazati pretplatu slanjem odgovarajućeg paketa (*unsubscribe packet*) koji sadrži identifikator i listu tema s kojih se klijent želi odjaviti. Kada posrednik primi poruku odjave, odgovara *Unsuback* paketom s istim identifikatorom kako bi potvrdio uspješnost odjave pretplate. U sljedećem odlomku bit će govora o temama i njihovoj ulozi usmjerivača unutar protokola MQTT.

1.4. MQTT tema

Tema je niz znakova u UTF-8 formatu koje posrednik koristi kako bi filtrirao poruke i točno ih proslijedio odgovarajućim klijentima. Teme se sastoje od jedne ili više razina koje su hijerarhijski odvojene kosom crtom prema naprijed (slika 11).



Slika 11. *Primjer MQTT teme* [7]

Svaka tema mora sadržavati barem jedan znak, a imena razina mogu sadržavati razmak i razlikuju velika i mala slova. Klijent ne treba stvoriti temu prije objave poruke jer će ju posrednik automatski stvoriti ako ne postoji u trenutku slanja poruke. U sustavu imenovanja MQTT tema postoje i dva zamjenska znaka koja klijentima olakšavaju slanje pretplata na razne teme:

1. Višerazinski zamjenski znak (*eng. multi-level wildcard*) – znak '#' predstavlja zamjenu za bilo koji broj hijerarhijski podređenih tema. Ako se klijent pretplati na temu sport/tenis/#, primat će obavijesti s teme sport/tenis i svih njenih podtema kao što su npr. sport/tenis/podloga, sport/tenis/igrač , sport/tenis/podloga/turnir itd. Treba napomenuti da znak # mora biti zadnji znak u imenu teme i mora biti odvojen separatorom /.

2. Jednorazinski zamjenski znak (eng. single-level wildcard) – znak '+' predstavlja zamjenu za jednu hijerarhijsku razinu teme. U pretplatu na temu sport+/igrač uključene su teme spor/tenis/igrač, sport/košarka/igrač itd. Znak '+' mora biti odvojen separatorom '/', a može se kombinirati sa znakom #.

Kod imenovanja tema klijenti su potpuno slobodni u biranju imena uz jednu iznimku koja se pojavljuje kod tema koje počinu znakom '\$'. Te teme se tretiraju posebno i ne spadaju u općenite pretplate, već su rezervirane za praćenje statistike MQTT posrednika. Uobičajena praksa je da teme za praćenje statistike počinju nizom znakova "\$SYS\". Sljedeća dva poglavlja bit će usmjerena na analizu postojeće programske podršku za protokol MQTT u programskom jeziku Java i MQTT posrednike otvorenog koda.

2. Programska podrška za protokol MQTT

Kako bi protokol MQTT mogao biti implementiran u aplikaciji, bilo je potrebno pronaći odgovarajuću programsku podršku i uključiti ju u izgradnju projekta. Programska podrška dostupna je u obliku klijentskih biblioteka koje nude sučelje za spajanje klijenata i posrednika MQTT te omogućuju korištenje specifičnih funkcija protokola MQTT opisanih u prošlom poglavlju. Popis svih klijentskih biblioteka za programski jezik Java može se pronaći na službenoj stranici protokola MQTT [8].

Eclipse Paho [9] klijentska biblioteka uključena je u aplikaciju zbog najkvalitetnije dokumentacije i najvećeg broja implementacijskih primjera na Webu. Ova MQTT klijentska biblioteka napisana je u Javi i omogućava razvoj aplikacija koje se izvode na Javinom virtualnom stroju (JVM, eng. Java Virtual Machine) ili drugim java-kompatibilnim platformama kao što je Android. Osim Eclipse Paho programske podrške kao stabilne klijentske biblioteke navode se XenQTT i Fusesource mqtt-client. Sve tri navedene biblioteke nude vrlo sličnu implementacijsku podršku za MQTT klijente u Javi. Uz uključivanje navedenih biblioteka moguće je koristiti dvije različite implementacije klijenata, sinkronu i asinkronu. Kod sinkronih klijenata svaka metoda za komunikaciju s posrednikom MQTT blokira druge operacije sve dok ne završi s izvođenjem, dok kod asinkronih klijenata te metode ne blokiraju ostale operacije koje se mogu izvršavati u pozadini [9]. XenQTT biblioteka uz implementaciju klijenta nudi i jednostavnu implementaciju posrednika MQTT (eng. *Mock Broker*) koji služi za lakše testiranje MQTT klijenata i same aplikacije.

3. Posrednici MQTT otvorenog koda

Uz odgovarajuću programsku podršku za protokol bilo je potrebno istražiti i posrednike MQTT otvorenog koda koji omogućavaju uspostavu MQTT komunikacije u Android aplikaciji. Posrednik MQTT zadužen je za prosljeđivanje objavljenih poruka i praćenje pretplata, on povezuje krajnje čvorove (klijente) aplikacije i vrši usporedbu objava i pretplata. Najrašireniji posrednici MQTT posrednici otvorenog koda su Mosquitto, HiveMQ i RabbitMQ.

Posrednik Mosquitto verzije 1.4.2 je vrlo jednostavan za konfiguraciju i pokretanje na vlastitom računalu, a otvoren je za uporabu na različitim platformama (Windows, Mac OS X, Linux itd.). Ovaj posrednik podržava protokole MQTT verzija 3.1 i 3.1.1 te nudi vrlo sličnu funkcionalnost kao i IBM-ov posrednik Real Small Message Broker (RSMB) koji je također dostupan za korištenje, ali nije otvorenog koda. Zbog kompatibilnosti Mosquitto i Eclipse projekata posrednik Mosquitto korišten je u aplikaciji zajedno s Eclipse Paho klijentskom bibliotekom. Još jedna prednost Mosquitto-a je implementacija dvije vrste klijenata, `mosquitto_pub` koji nudi mogućnost objave informacija te `mosquitto_sub` koji nudi mogućnost pretplate na informacije od interesa.

Za razliku od posrednika Mosquitto, posrednik HiveMQ verzije 2.3.1 ne nudi implementaciju klijenata preko komandne linije, ali zato nudi klijentsko web sučelje za povezivanje klijenata i posrednika preko web priključnica (*websockets*). Ovaj posrednik također je platformno neovisan i nudi opciju dodavanja proširenja (*eng. plugin*) osnovnih funkcionalnosti protokola.

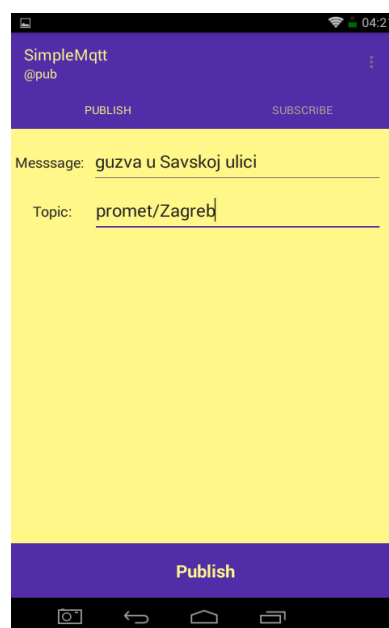
Još jedan posrednik kojeg valja spomenuti je RabbitMQ, on za razliku od prethodna dva posrednika nije namjenjen isključivo protokolu MQTT te podržava nekoliko protokola za razmjenu poruka kao što su AMQP, STOMP i HTTP. Svaki od ovih protokola moguće je koristiti uključivanjem odgovarajućih proširenja prilikom pokretanja posrednika. RabbitMQ podržava MQTT protokol verzije 3.1 i većinu njegovih funkcionalnosti, ali nema ugrađenu podršku za implementaciju klijenata te ne nudi klijentsko sučelje za testiranje aplikacija. Treba još napomenuti da ovaj posrednik ne podržava slanje poruka s QoS=2.

4. O razvijenoj Aplikaciji

Razvijena aplikacija za dojavu događaja na uređajima s operacijskim sustavom Android uz implementaciju protokola MQTT predstavlja krajnji cilj završnog rada. Ona omogućava krajnjim korisnicima jednostavnu, brzu i ekonomičnu komunikaciju slanjem pretplata i objava prema posredniku MQTT koji prosljeđuje poruke i pamti pretplate te predstavlja središnju točku MQTT komunikacije. Kako bi u aplikaciji bila omogućena razmjena poruka na principu objavi-pretplati korišten je posrednik Mosquitto uz Eclipse Paho klijentsku biblioteku pisanu u programskom jeziku Java. Mosquitto posrednik omogućava MQTT klijentima slanje i primanje poruka bez poznavanja primatelja odnosno pošiljatelja što olakšava i ubrzava komunikaciju te ju čini pogodnom za mobilne uređaje s ograničenim resursima poput života baterije i propusnosti informacija (*bandwidth*).

4.1. Dizajn Sustava

Sučelje aplikacije sastoji se od dvije klizne kartice (*sliding tabs*) za izradu pretplata (*subscribe*) i objava (*publish*). Kartica za objavu sastoji se od dva polja za unos teksta poruke i teme preko koje posrednik saznaje kome treba isporučiti poslanu poruku (slika 12).



Slika 12. *kartica objave*

Nakon što korisnik ispuni navedena polja pritiskom na tipku *Publish* poruka se šalje posredniku koji na osnovu zapamćenih pretplata na teme prosljeđuje poruku svim pretplaćenim korisnicima. Kliznim pokretom prstom po ekranu korisnik može lako prijeći na drugu karticu za pretplatu koja sadrži jedno polje za unos teme (slika 13).



Slika 13. *kartica pretplate*

Nakon unosa teme i pritiska na tipku *Subscribe* posrednik prima i sprema pretplatu na navedenu temu koja će mu kasnije poslužiti kao informacija za pravilno prosljeđivanje poruka na osnovu usporedbe pretplata i objava. Kada se uspješno pretplati na temu, korisnik može u bilo kojem trenutku primiti poruku. Sve primljene poruke spremaju se u internu memoriju uređaja kako bi ih korisnik mogao pregledati u bilo kojem trenutku.

Pregled poruka omogućen je u posebnom prozoru do kojeg korisnik dolazi pritiskom akcijske tipke u gornjem desnom kutu alatne trake i odabirom polja *Messages* (slika 14).



Slika 14. *Prozor za pregled poruka*

4.2. Tehničke značajke

Za izradu aplikacije korištena je objektno usmjerena arhitektura programskog jezika Java (jdk 1.8.0_20) uz Android programski okvir Android Tools (21.1.2) unutar Android Studio razvojnog okruženja verzije 1.0. koje je namijenjeno i specijalizirano za razvoj i testiranje Android aplikacija. Android Studio nudi mnoge dodatke i proširenja koja olakšavaju izradu same aplikacije. Za pokretanje, testiranje i nadogradnju aplikacije zadužen je Gradle Build sustav (Gradle 2.2.1) koji generira izvršnu Android *.apk* datoteku za pokretanje aplikacije na mobilnom uređaju. Ovaj sustav sadrži *build.gradle* datoteku s konfiguracijskim opcijama specifičnim za izgradnju Android aplikacije i deklaracijama svih ovisnosti (*eng. dependencies*) koje se uključuju prilikom prevođenja (slika 15).

```

apply plugin: 'com.android.application'

android {
    compileSdkVersion 21
    buildToolsVersion "21.1.2"

    defaultConfig {
        applicationId "hr.fer.zavrsni_rad.mqtt_implementation"
        minSdkVersion 15
        targetSdkVersion 21
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(include: ['*.jar'], dir: 'libs')
    compile 'com.android.support:appcompat-v7:21.0.3'
    compile 'it.neokree:MaterialTabs:0.11'
    compile 'com.android.support:recyclerview-v7:+'
    compile files('libs/org.eclipse.paho.client.mqttv3-1.0.2.jar')
}

```

Slika 15. *build.gradle datoteka aplikacije*

Kako bi biblioteke bile uključene u izgradnju projekta moraju se nalaziti unutar dependencies elementa koji kod razvijene aplikacije sadrži podršku Android biblioteka *com.android.support:appcompat-v7* i *com.android.support:recyclerview-v7*. Prva biblioteka omogućava korištenje razreda ActionBar za implementaciju alatne trake, a druga korištenje razreda RecyclerView za učinkovito recikliranje elemenata kolekcija (*npr. liste*) unutar sučelja. Obje biblioteke predstavljaju proširenja za Android uređaje s razinom API-ja većom od 7. Proširenje za izradu kliznih kartica omogućena su uključivanjem biblioteke *it.neokree:MaterialTabs*, a programska podrška za protokol MQTT osigurana je uključivanjem *org.eclipse.paho.client.mqttv3-1.0.2* klijentske biblioteke iz *libs* direktorija.

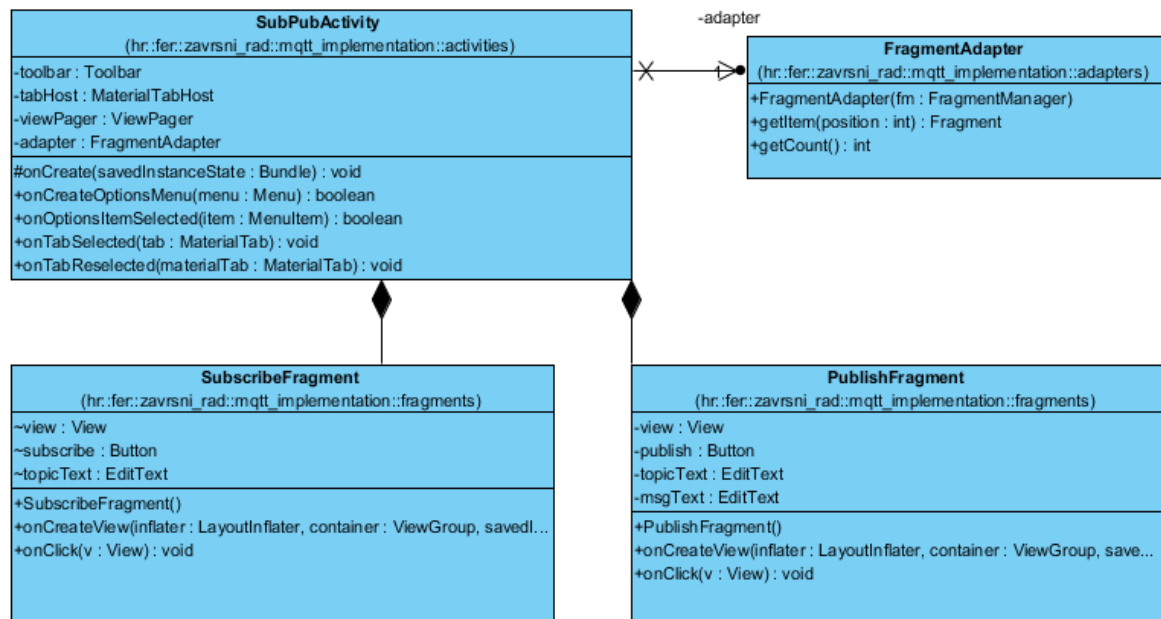
4.3. Programske značajke

Aplikacija se sastoji od dva prozora (*Activity*), glavnog prozora **SubPubActivity** za objavu i pretplatu koji se otvara pri pokretanju i prozora za prikaz primljenih poruka **MessageActivity**. Kako bi se prozori mogli koristiti treba ih deklarirati u *AndroidManifest.xml* datoteci koja Android sustavu daje ključne informacije vezane uz aplikaciju i omogućava mu pokretanje njenog koda (slika 16).

```
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name=".activities.SubPubActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity
        android:name=".activities.MessageActivity"
        android:label="" />
    <service
        android:name=".services.MQTTService"
        android:enabled="true" >
    </service>
</application>
```

Slika 16. dio *AndroidManifest.xml* datoteke s informacijama vezanim uz aplikaciju

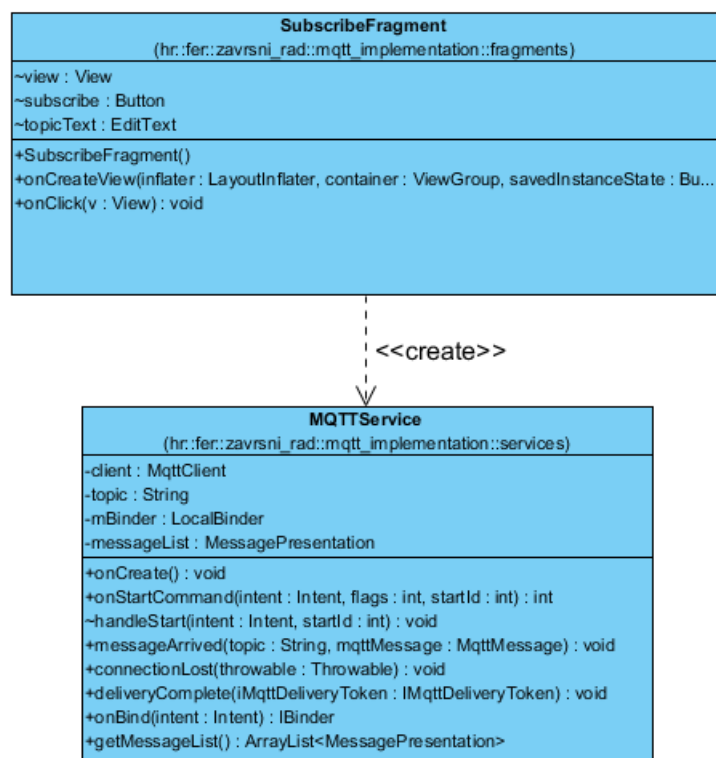
Prozor **SubPubActivity** u manifest datoteci je deklariran kao glavni prozor (action.MAIN) koji se stvara pri pokretanju (category.LAUNCHER) aplikacije. On se sastoji od dva Fragmenta, za objavu poruka **PublishFragment** i za slanje pretplata **SubscribeFragment** (Slika 17). Ta dva fragmenta izmjenjuju se uporabom kliznih kartica i razreda ViewPager unutar prozora **SubPubActivity**. ViewPager je razred koji uz pomoć **FragmentAdaptora** izmjenjuje pozicije fragmenata kliznim pokretom po ekranu mobitela. Za promjenu tabova sukladno promijeni fragmenata zadužen je razred MaterialTabHost koji je uključen u projekt preko proširenja *it.neokree:MaterialTabs* za izradu kliznih kartica. Kako bi simultano mijenjanje fragmenata i tabova bilo omogućeno mora se uspostaviti komunikacija između razreda ViewPager i MaterialTabHost. Komunikacija se ostvaruje postavljanjem promatrača koji osluškuju (*listener*) promjene pozicija fragmenata i tabova te mijenjaju izgled sučelja sukladno tim promjenama.



Slika 17. *Dijagram razreda za SubPubActivity prozor s dva fragmenta u obliku kliznih kartica*

Fragmenti **SubscribeFragment** i **PublishFragment** korisniku nude sučelje za objavu poruka i slanje pretplata posredniku. Kada korisnik želi objaviti poruku mora u polja za unos teksta navesti odgovarajuću temu i poruku koju želi poslati. Slanje poruke nije trajna operacija pa se može izvesti direktno u fragmentu unutar **SubPubActivity** prozora nakon pritiska na tipku *Publish*. Za razliku od slanja poruke, pretplaćivanje na temu je trajna operacija koja mora konstantno održavati i oslušivati vezu s posrednikom te u bilo kojem trenutku, neovisno o stanju aplikacije, mora biti spremna za primanje proslijeđene poruke. Zbog toga se ova operacija ne može izvesti direktno unutar **SubPubActivity** prozora koji je kratkog životnog vijeka (short-lived), točnije aktivan je samo kada korisnik ima upaljenu aplikaciju i otvoren baš taj prozor što znači da bi TCP/IP konekcija s MQTT posrednikom trajala sve dok korisnik ne promijeni aktivan prozor ili izađe iz aplikacije.

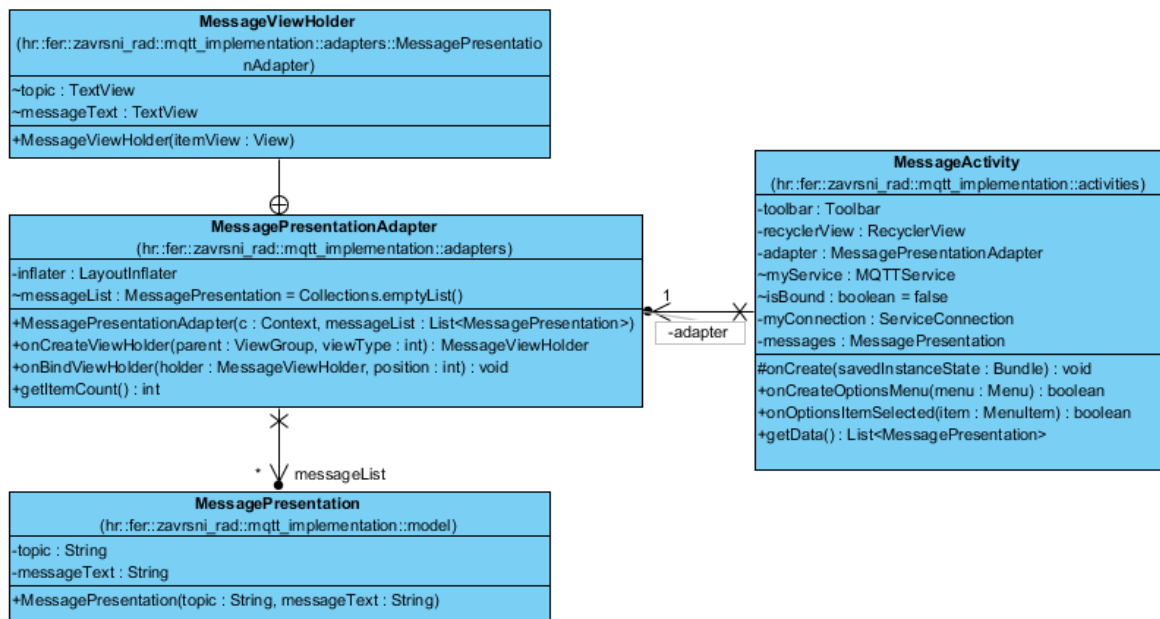
Kako bi trajna konekcija bila omogućena neovisno o stanju aplikacije i trenutno aktivnom prozoru korišten je servis **MQTTService** koji je, za razliku od prozora **SubPubActivity**, pogodan za implementaciju duže živućih (*long-running*) operacija. MQTTService pokreće se iz fragmenta **SubscribeFragment** (slika 18). pritiskom tipke *Subscribe* i cijelo vrijeme je aktivan u pozadini što ga čini idealnim za uspostavu trajne TCP/IP konekcije s posrednikom.



Slika 18. stvaranje razreda *MQTTService* iz *SubscribeFragmenta*

Razred **MQTTService** zadužen je za stvaranje MQTT klijenta, dodavanje novih pretplata i održavanje trajne veze s posrednikom. Kada posrednik preko veze proslijedi poruku klijentu, **MQTTService** dodat će ju u listu poruka koju će zatim pohraniti u internu memoriju uređaja.

Prozor **MessageActivity** zadužen je za prikaz svih poruka i njihovih tema. MQTT poruke određene su razredom **MessagePresentation** koji za svaku poruku definira temu i sadržaj. Kada korisnik otvori prozor **MessageActivity**, on iz interne memorije čita listu MQTT poruka (primjeraka razreda **MessagePresentation**) i prikazuje sadržaj svake poruku i njene teme na ekranu. Za prikaz liste poruka koristi se razred RecyclerView koji je uključen u projekt preko proširenja *com.android.support:recyclerview-v7*. RecyclerView uz primjenu obrasca ViewHolder pruža uslugu recikliranja odnosno dohvata već izgrađenih instanci sučelja iz memorije što ubrzava navigaciju po listi poruka i smanjuje memorijsko opterećenje (slika 19).

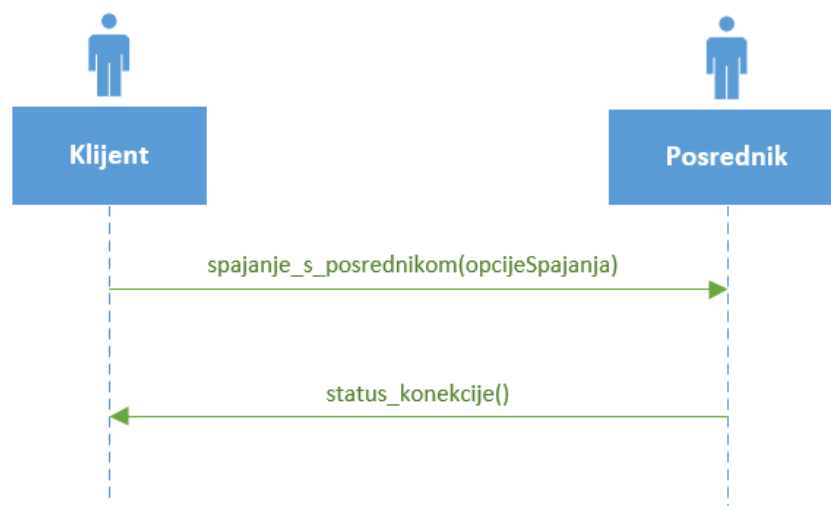


Slika 19. dijagram razreda za prikaz poruka unutar prozora MessageActivity uz korištenje ViewHolder obrasca

4.4. Funkcionalni zahtjevi

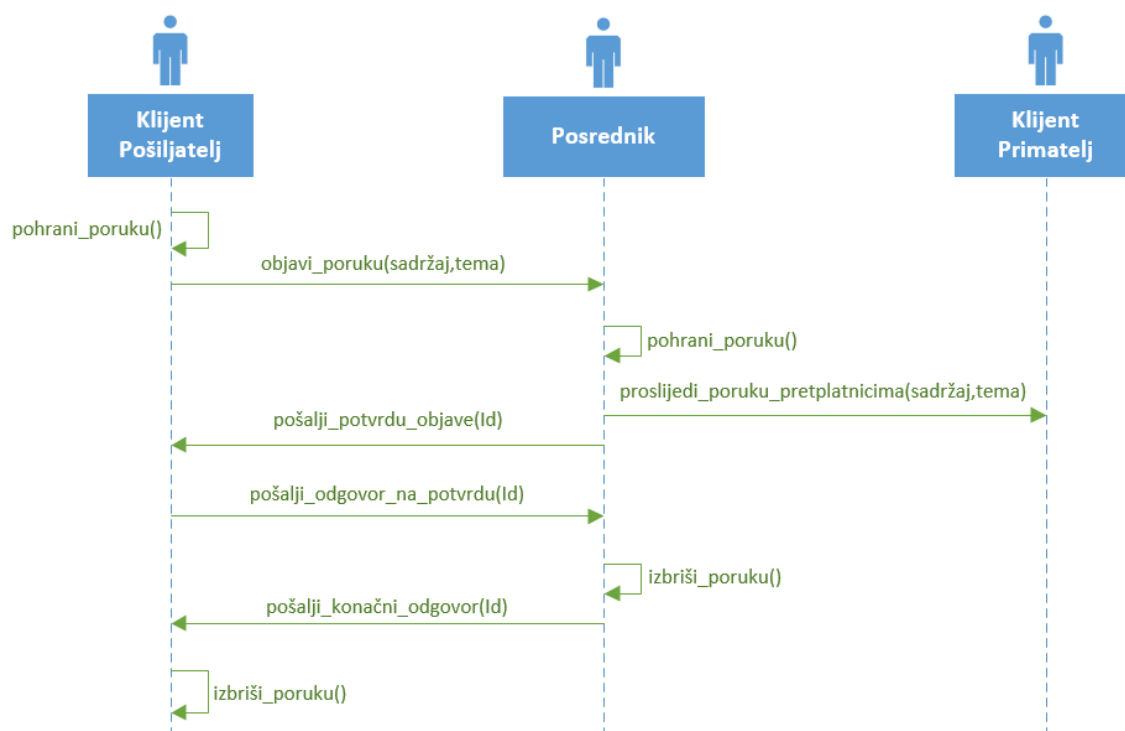
Osnovna zadaća razvijene Android aplikacije je povezivanje klijenata protokolom MQTT uz posrednikovu pomoć. Ona mora omogućiti učinkovito spajanje i odspajanje klijenata s posrednikom MQTT, objavljivanje, pretplaćivanje i isporučivanje poruka te prikaz liste isporučenih poruka u za to predviđenom prozoru.

Ono što prethodi svakoj vrsti komunikacije je spajanje klijenta i posrednika. Klijent inicira početak komunikacije i opcionalno postavlja zastavice spajanja (*MqttConnectOptions*), a posrednik odgovara porukom koja nosi informaciju o uspješnosti povezivanja (slika 20).



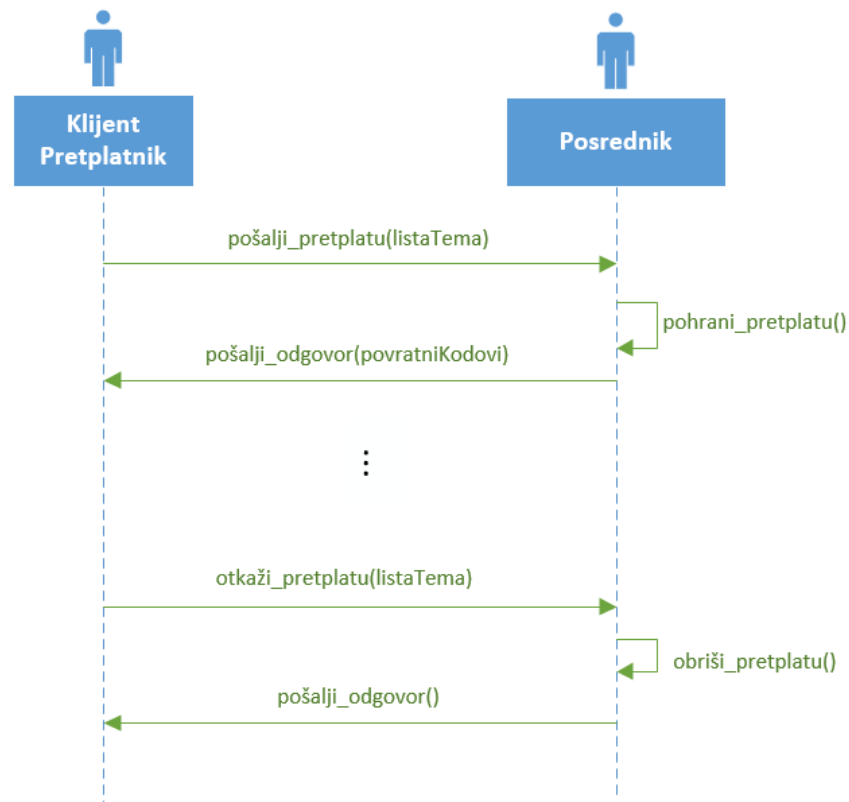
Slika 20. uspostava veze između klijenta i posrednika

Nakon uspješno uspostavljene konekcije s posrednikom, klijentu je omogućena objava poruka i izrada pretplata. Kada želi objaviti poruku, on šalje posredniku njen sadržaj i puni naziv teme te postavlja razinu kvalitete servisa (QoS). Za sigurnu isporuku poruke točno jednom koristi se QoS vrijednosti 2 kod koje posrednik, nakon spremanja i prosljeđivanja poruke svim pretplatnicima, vraća pošiljatelju potvrdu o njenoj uspješnoj objavi. Kako bi se izbjeglo slanje dupliciranih poruka pošiljatelj posredniku odgovara na potvrdu koji tada briše pohranjenu poruku i vraća konačni odgovor (slika 21).



Slika 21. Objava i prosljeđivanje poruke

Kada se želi pretplatiti na temu, klijent šalje naziv teme koji može sadržavati i posebne znakove „#“ i „+“ korištene za izradu pretplata na više tema odjednom. Nakon što primi listu pretplata, posrednik ju pamti, kako bi kasnije znao koje poruke mora proslijediti klijentu, i šalje odgovor sa statusnim kodom za svaku pretplatu iz primljene liste pretplata. Klijent može i otkazati pretplatu slanjem odgovarajuće poruke posredniku koji zatim vraća odgovor o uspješnosti otkazivanja (slika 22).



Slika 22. *izrada i otkazivanje pretplata*

4.5. Upute za instalaciju

Instalacija aplikacije na uređaj s operacijskim sustavom Android moguća je u nekoliko vrlo jednostavnih koraka:

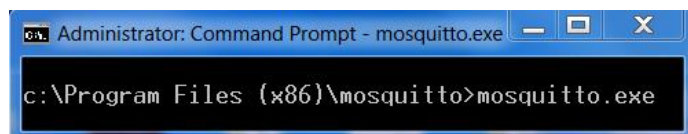
1. U glavnom izborniku uređaja treba odabrati stavku opcije/sigurnost (*settings/security*) i omogućiti instaliranje aplikacija s nepoznatih izvora (*unknown sources*).
2. Priključiti uređaj USB kablom na računalo i uključiti USB skladište (*storage*) za kopiranje datoteka s računala na SD karticu Android uređaja.
3. Na računalu unutar matičnog projekta u poddirektoriju *app/build/outputs/apk* pronaći *app-debug.apk* datoteku i kopirati ju u proizvoljni direktorij unutar USB skladišta dostupnog na računalu.
4. Sigurno odspojiti Android uređaj s računala.
5. Korištenjem upravitelja datotečnog sustava (*file manager*) uređaja potrebno je pronaći direktorij u koji je kopirana *app-debug.apk* datoteka s računala, otvoriti datoteku te instalirati i potom pokrenuti aplikaciju.

Kako bi aplikacija mogla komunicirati preko protokola MQTT potrebno ju je povezati s odgovarajućim posrednikom. Pokretanje i konfiguracija posrednika MQTT na vlastitom računalu detaljnije će biti objašnjeni u sklopu sljedećeg poglavlja.

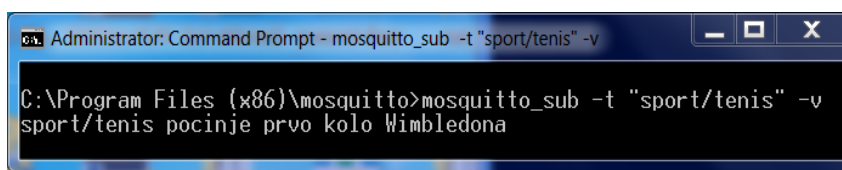
4.6. Konfiguriranje poslužitelja za razvijenu aplikaciju

U 4. Poglavlju dan je pregled posrednika MQTT otvorenog koda koji aplikaciji omogućavaju korištenje funkcionalnosti protokola MQTT. Sva tri prethodno predstavljena posrednika, Mosquitto, HiveMQ i RabbitMQ, mogu se jednostavno pokrenuti na vlastitom računalu i nude različite mogućnosti promatranja prometa komunikacije posrednika s klijentima.

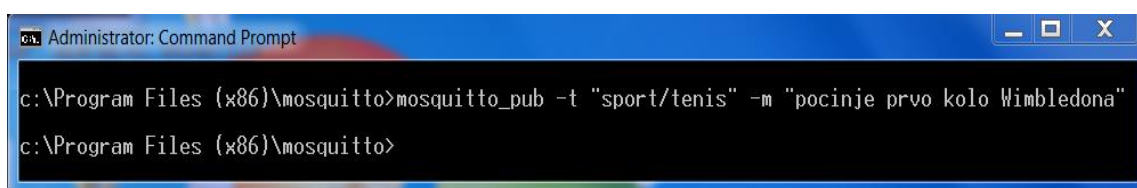
Posrednik Mosquitto vrlo je popularan jer, uz osnovne funkcionalnosti protokola, nudi i implementacije klijenata pošiljatelja i primatelja (mosquitto_pub i mosquitto_sub). Oba klijenta, kao i sam posrednik, mogu se pokrenuti iz komandne linije što je vrlo pogodno za testiranje aplikacije (slika 12, 13 i 14).



Slika 12. *Pokretanje Mosquitto posrednika iz komandne linije*



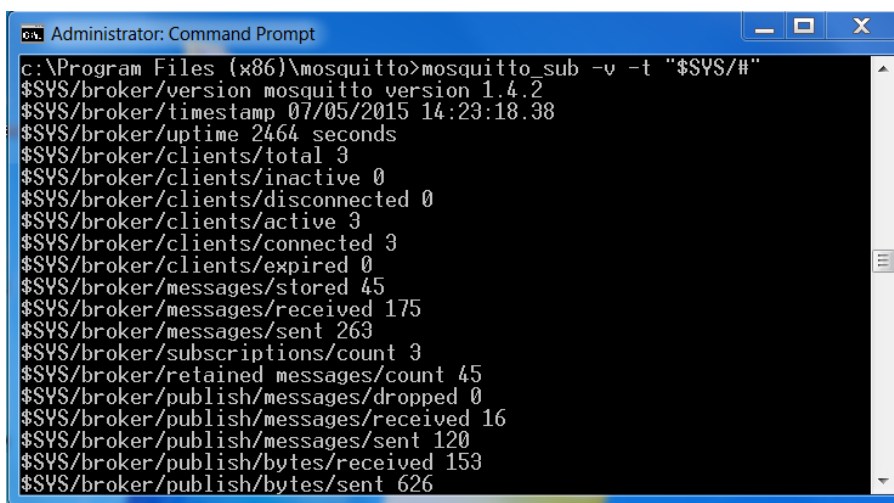
Slika 13. *Pokretanje mosquitto_sub pretplatnika*



Slika 14. *Pokretanje mosquitto_pub izdavača*

Nakon pokretanja Mosquitto posrednika (slika 12) omogućeno je njegovo povezivanje s klijentima te pretplata i objava poruka. Ako se želimo pretplatiti na temu pomoću ugrađene Mosquitto podrške, pokrećemo mosquitto_sub klijenta i opcijom -t odabiremo temu, a opcijom -v omogućavamo prikaz poruke s imenom teme i njenim sadržajem. Mosquitto_sub klijent pokrenut je trajno i cijelo vrijeme čeka na poruke koje mu šalje posrednik.

Za objavu poruke pokrećemo `mosquitto_pub` klijenta i opcijom `-t` odabiremo temu na koju želimo objaviti poruku, a opcijom `-m` definiramo sadržaj objavljene poruke. Nakon što je poruka objavljena `mosquitto_pub` izdavač završit će s izvođenjem, a Mosquitto posrednik će na temelju usporedbe objave i pretplata znati kojim klijentima treba proslijediti poslanu poruku. Klijenti se mogu pretplatiti i na teme vezane uz statistike i informacije o posredniku, te teme počinju nizom znakova `"$SYS/"`. Primjer jedne takve pretplate iz komandne linije je `mosquitto_sub -v -t "$SYS/#"` kojom se klijent se pretplaćuje na sve informacije o posrednika (slika 15).



```
Administrator: Command Prompt
c:\Program Files (x86)\mosquitto>mosquitto_sub -v -t "$SYS/#"
$SYS/broker/version mosquitto version 1.4.2
$SYS/broker/timestamp 07/05/2015 14:23:18.38
$SYS/broker/uptime 2464 seconds
$SYS/broker/clients/total 3
$SYS/broker/clients/inactive 0
$SYS/broker/clients/disconnected 0
$SYS/broker/clients/active 3
$SYS/broker/clients/connected 3
$SYS/broker/clients/expired 0
$SYS/broker/messages/stored 45
$SYS/broker/messages/received 175
$SYS/broker/messages/sent 263
$SYS/broker/subscriptions/count 3
$SYS/broker/retained messages/count 45
$SYS/broker/publish/messages/dropped 0
$SYS/broker/publish/messages/received 16
$SYS/broker/publish/messages/sent 120
$SYS/broker/publish/bytes/received 153
$SYS/broker/publish/bytes/sent 626
```

Slika 15. Pretplata na sve informacije vezane uz Mosquitto posrednika

Posrednik HiveMQ za povezivanje klijenata preko web priključnica nudi klijentsko web sučelje. Na MS Windows operacijskom sustavu posrednik se pokreće otvaranjem `run.bat` datoteke u `bin` direktoriju. Za spajanje klijenata s posrednikom preko HiveMQ klijentskog web sučelja moramo u `configuration.properties` datoteci omogućiti korištenje web priključnica postavljanjem `websockets.enabled` atributa na vrijednost `true` i podešavanjem porta web priključnica na vrijednost 8000. Kada postavimo navedene vrijednosti klijenti će se preko sučelja na internetu moći spajati s posrednikom pokrenutim na našem vlastitom računalu (slika 16) te preko njega objavljivati, primati i pregledavati MQTT poruke (slika 17).

Connection ● disconnected ⌵

Host: Port: ClientID: Connect

Username: Password: Keep Alive: SSL: ☐ Clean Session: ☒

Slika 16. *Spajanje s MQTT posrednikom pokrenutim na vlastitom računalu preko web priključnice*

Connection ● connected ⌵

Publish ⌵

Topic: QoS: Retain: ☐ Publish

Message:

Subscriptions ⌵

Add New Topic Subscription

Qos: 2 ☒ X

sport/#

Messages ⌵

Time	Topic	QoS	Message
2015-06-08 10:09:47	sport/tenis	Qos: 1	wimbledon je počeo

Slika 17. *Izgled web sučelja za pretplatu i objavi MQTT poruka preko HiveMQ posrednika*

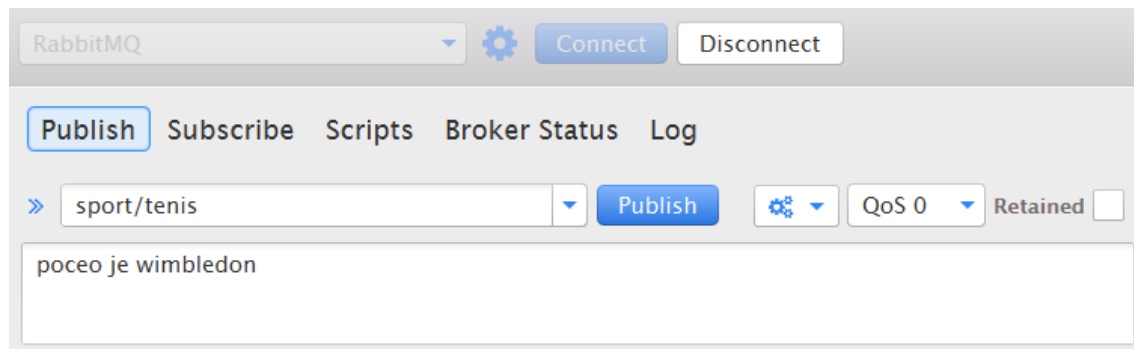
HiveMQ nudi i opciju dodavanja proširenja (eng. *plugin*) osnovnih funkcionalnosti posrednika. Jedno od proširenja je i proširenje za prijavu poruka (eng. *MQTT Message Log*) koje omogućuje posredniku da objavljuje poruke vezane uz spajanje i odspajanje, objavljivanje i pretplaćivanje te otkazivanje pretplata klijenata (slika 18) što je korisno prilikom otklanjanja grešaka i testiranja aplikacija. Proširenja dodajemo kopiranjem njihove *.jar* datoteke u *plugins* direktorij HiveMQ posrednika.

```

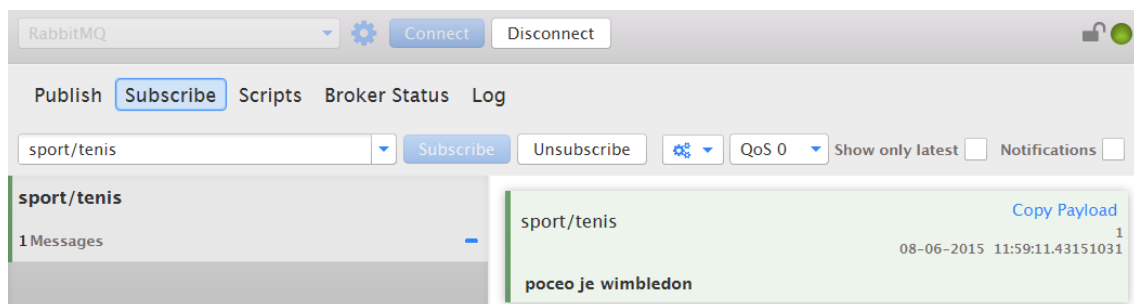
Administrator: Command Prompt - run
2015-06-08 10:21:53,233 INFO - Started HiveMQ 2.3.1 in 2961ms
2015-06-08 10:22:04,738 INFO - Client Borna Sirovica connected
2015-06-08 10:22:07,490 INFO - Subscribe from client Borna Sirovica received: sport/# QoS: 2
2015-06-08 10:22:09,197 INFO - Client Borna Sirovica sent a message to topic "sport/tenis": "wimbledon je počeo" (QoS: 1, retained: false)
2015-06-08 10:22:14,025 INFO - Unsubscribe from client Borna Sirovica received: sport/#
2015-06-08 10:23:01,493 INFO - Client Borna Sirovica sent DISCONNECT message
  
```

Slika 18. *HiveMQ posrednik pokrenut na vlastitom računalu s proširenjem za prijavu (logiranje) poruka vezanih za aktivnost klijenata*

Posrednik RabbitMQ nudi implementaciju većeg broja komunikacijskih protokola pa je za implementaciju protokola MQTT potrebno pokrenuti naredbu *rabbitmq-plugins enable rabbitmq_mqtt* koja omogućava MQTT adapter unutar posrednika. Za razliku od prethodna dva posrednika RabbitMQ ne nudi podršku za implementaciju MQTT klijenata tako da se za testiranje aplikacije može koristiti neki od javno dostupnih MQTT klijentskih alata kao što je MQTT.fx koji nudi klijentsko sučelje za komunikaciju protokolom MQTT (slika 19 i 20).



Slika 19. *Objava poruka na MQTT.fx klijentskom sučelju preko posrednika RabbitMQ*



Slika 20. *Pretplata i pregled poruka na MQTT.fx klijentskom sučelju preko posrednika RabbitMQ*

Zaključak

Danas, kada se sve više govori o internetu stvari (*Internet of Things, IoT*) i samostalnoj komunikaciji uređaja (Machine to Machine, M2M) preko interneta, protokol MQTT, zbog svoje usklađenosti s ograničenjima i zahtjevima ovakve vrste slobodne i dinamične razmjene podatka, postaje jedan od najzanimljivijih i najperspektivnijih komunikacijskih protokola. Kako broj jednostavnih uređaja s ograničenim resursima koji imaju mogućnost povezivanja internetom raste velikom brzinom, raste i potreba za što lakšom, štedljivijom, otpornijom i pouzdanijom komunikacijom koju nudi ovaj protokol sveprisutan u fizičkom svijetu senzora, aktuatora, telefona i tableta.

Arhitektura objavi-pretplati, korištena u razvijenoj aplikaciji, pokazala se vrlo pogodnom za povezivanje mobilnih uređaja. Ona nudi anonimnu komunikaciju između klijenata i omogućava im vrlo jednostavnu razmjenu podataka uz malo opterećenje na život baterije i mrežni promet. Kako bi klijentski dio aplikacije što manje utjecao na performanse mobilnog uređaja većina odgovornosti svaljena je na posrednika protokola MQTT koji preuzima poruke i brine o njihovoj pravilnoj isporuci.

Navedene kvalitete i prednosti protokola MQTT čine ga otvorenim i pogodnim za daljnji razvoj i prilagodbu na uređajima niske propusnosti s ograničenim mogućnostima obrade i procesorske moći. Pošto je komunikacija među tim uređajima sve raširenija i naprednija, protokol MQTT bit će sigurno jedan od protokola budućnosti.

Literatura:

- [1] HiveMQ, „MQTT 101 – How to Get Started with the lightweight IoT Protocol“, <http://www.hivemq.com/how-to-get-started-with-mqtt/>
- [2] Peter R. Egli (2013.), „MQTT - MQ Telemetry Transport for Message“, <http://www.slideshare.net/PeterREgli/mq-telemetry-transport>
- [3] Stephen Nicholas (2012.), „Power Profiling: HTTPS Long Polling vs. MQTT with SSL, on Android“, <http://stephendnicholas.com/archives/1217>
- [4] HiveMQ, „MQTT Essentials Part 3: Client, Broker and Connection Establishment“, <http://www.hivemq.com/mqtt-essentials-part-3-client-broker-connection-establishment/>
- [5] Oasis (2014), „MQTT Version 3.1.1 OASIS Standard“, <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>
- [6] HiveMQ, „MQTT Essentials Part 4: MQTT Publish, Subscribe & Unsubscribe“, <http://www.hivemq.com/mqtt-essentials-part-4-mqtt-publish-subscribe-unsubscribe/>
- [7] HiveMQ, „MQTT Essentials Part 5: MQTT Topics & Best Practices“, <http://www.hivemq.com/mqtt-essentials-part-5-mqtt-topics-best-practices/>
- [8] MQTT community wiki, <https://github.com/mqtt/mqtt.github.io/wiki>
- [9] Eclipse Paho, Java Client documentation , <http://www.eclipse.org/paho/files/javadoc/index.html>

Aplikacija za dojavu događaja na uređajima s operacijskim sustavom Android

Sažetak

Protokol MQTT (*Message Queue Telemetry Transport*) je otvoren, jednostavan i lagan protokol koji omogućava raspodijeljenu razmjenu poruka između krajnjih čvorova na principu objavi-pretplati. Pogodan je za komunikaciju uređaja s ograničenim resursima poput kratkog života baterije, male mrežne propusnosti i memorijskog kapaciteta te visoke latencije. Središnju ulogu u MQTT komunikaciji ima posrednik koji gospodari razmjenom podataka prosljeđivanjem objava i pamćenjem pretplata. Klijenti komuniciraju isključivo preko posrednika s kojim održavaju trajnu TCP/IP vezu i razmjenjuju poruke. Kako bi aplikacija mogla implementirati protokol MQTT u projekt treba uključiti jednu od javno dostupnih klijentskih biblioteka i povezati ju s posrednikom MQTT otvorenog koda. Razvijena Android aplikacija nudi korisniku jednostavno sučelje za uspostavu konekcije s posrednikom, objavu poruka, slanje pretplata i pregleda pridošlih poruka. Protokol MQTT najviše se veže uz sve popularniji pojam Interneta stvari i samostalnu komunikaciju uređaja preko interneta (M2M).

Ključne riječi: protokol MQTT, posrednik, objava, pretplata, MQTT tema, klijentske biblioteke, Eclipse Paho, Mosquitto, HiveMQ, Android aplikacija, Internet stvari.

Android Application for Event Notification

Abstract

MQTT (*Message Queue Telemetry Transport*) is open, light weight and simple protocol that allows distributed message exchange between communication end nodes based on publish/subscribe principle. This protocol is suitable for communication between devices with limited resources such as short battery life, low bandwidth, small memory capacity and high latency. Broker has a major role in MQTT communication, he controls data exchange with message forwarding and storing subscriptions. Clients communicate only over broker with whom they maintain a permanent TCP/IP connection. For application to implement MQTT protocol, project should include one of the publicly available client libraries and connect it with open source MQTT broker. Developed application offers a simple user interface for establishing connection to broker, publishing messages, sending subscriptions and reviewing newly arrived messages. MQTT protocol is often attached with increasingly popular concept of the Internet of Things (IoT) and Machine to Machine communication (M2M).

Keywords: protocol MQTT; broker; publish; subscribe; MQTT topic; client libraries; Eclipse Paho; Mosquitto; HiveMQ; Android application; Internet of Things