Универзитет у Београду
Математички факултет

Александра Бранковић 1057/2017

Употреба протокола *MQTT* у мобилним апликацијама

Мастер рад

Београд,
2019.

Универзитет у Београду

Мастер рад

Аутор:	Александра Бранковић 1057/2017
Наслов:	Употреба протокола *MQTT* у мобилним апликацијама
Чланови комисије:	др Мирослав Марић (Ментор) Математички факултет, Универзитет у Београду др Саша Малков Математички факултет, Универзитет у Београду др Александар Картељ Математички факултет, Универзитет у Београду

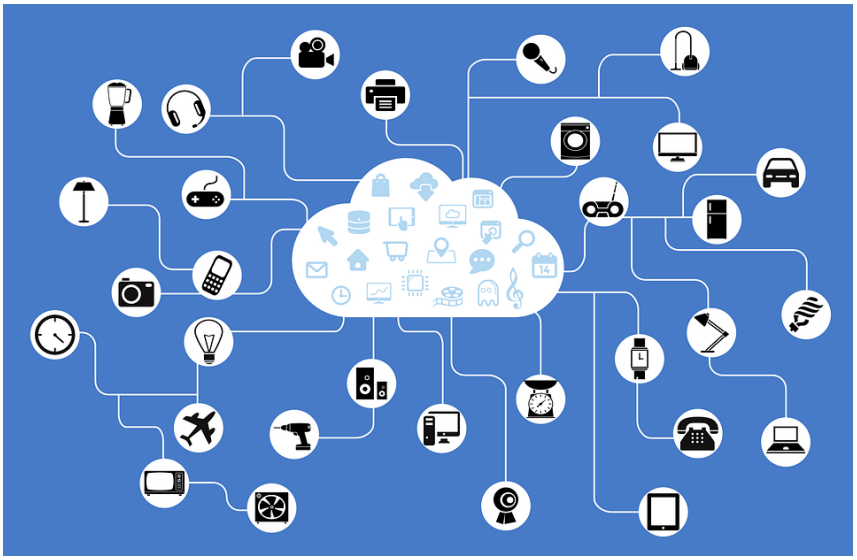
Датум одбране: _____

Садржај

Глава 1

Увод

Концепт Интернет ствари (енг. *Internet of Things – IoT*) представља умрежавање, комуникацију и надгледање уређаја уз помоћ интернета као медија комуникације, где уређаји представљају физичке објекте из свакодневне употребе, индустрије, телеметрије, итд (слика 1.1 [1] илустративно приказује разноврсност повезаних објеката). Будући да повезани уређаји могу бити ограничени (са слабим процесором, меморијом и батеријом), потребни су протоколи које би и такви уређаји могли да подрже.



Слика 1.1: Интернет ствари

[1] Слика је преузета са сајта https://pixabay.com

Протокол *Message Queing Telemetry Transport* (*MQTT*) је један од најпопуларнијих *IoT* протокола данашњице. Своју популарност је достигао захваљујући особинама попут: једноставности, могућности одржавања континуиране сесије, дефинисања више нивоа квалитета услуге под којима ће се поруке испоручити, итд.

Циљ рада је приказ карактеристика и начина функционисања протокола *MQTT*, као и поређење са другим протоколима. За потребе рада биће имплементирана мобилна апликација за чет (енг. *chat*) која ће у размени порука употребљавати протокол *MQTT*. У реализацији апликације биће коришћен алат *Xamarin.Forms* који омогућава развој вишеплатформских мобилних апликација. Апликација ће бити имплементирана за *Android* мобилне платформе са могућношћу надоградње за *iOS*.
У имплементацији апликације биће коришћена библиотека *M2Mqtt*, а улогу сервера ће обављати бесплатан и слободан *Mosquitto* сервер.

Поглавље 1 садржи основне информације о природи и намени рада. У поглављу 2 описани су општи концепти о протоколима и моделима комуникације. Затим је у поглављу 3 описана историја и интерна структура протокола *MQTT*. Поглавље 4 садржи информације о алатима и окружењима употребљеним у изради аликације. У поглављу 5 биће описана апликација за чет написана помоћу алата *Xamarin.Forms* и оквира *.NET*.

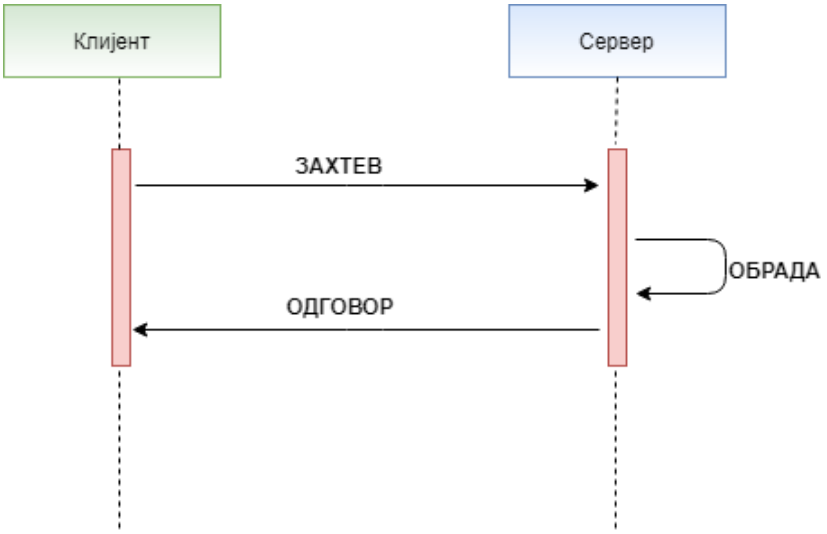
Глава 2

Протоколи и модели размене информација

Протокол представља скуп правила и смерница који одређују начин преноса информација.
Слој апликација, који представља највиши ниво *TCP/IP* и *OSI* референтног модела архитектуре мреже, дефинише апликативне протоколе који одређују механизме размена порука, безбедност података, скалабилност, интероперабилност, итд. док су слојеви испод (у архитектури мреже) задужени за успостављање комуникационих канала и исправно испоручивање података из слоја апликација (више о овоме се може наћи у [1]). Примери апликативних протокола су *HTTP, MQTT, FTP, XMPP*, итд.
Код размене података често постоје стране (у овом случају апликације дефинисане у слоју апликација) које захтевају услуге и оне стране које их пружају. Једна таква архитектура је позната као клијент - сервер архитектура где се клијент дефинише као страна која захтева услугу, а серверска страна као она која пружа услугу. У наставку ће се размена информација односити на клијент-сервер архитектуру.
Постоји више механизама, тј. начина на који долази до размена информација, међу којима су и *захтев-одговор* и *објава-претплата* модели.

2.1 Модел захтев-одговор

Код модела захтев-одговор (енг. *request-response*), једна страна издаје (шаље) захтев за одређеном информацијом, док друга страна шаље одговор. Најчешће је потребно да дође до низа оваквих интеракција да би процес размене порука био комплетиран. Овај модел се врло често користи у клијент - сервер архитектури (слика 2.1). Протокол *HTTP* користи овај модел комуникације, где клијент (углавном веб претраживач) издаје *HTTP* захтев серверу. Сервер, на коме су смештени веб садржаји, шаље одговор у виду *HTTP* одговора који садржи статусне информације о захтеву, а може и садржати и захтевани садржај. У протоколу *HTTP* клијент повлачи (енг. *pull*) информације са сервера, уместо да сервер гура информације ка клијенту. Захтев-одговор модел је пример комуникације један-на-један.



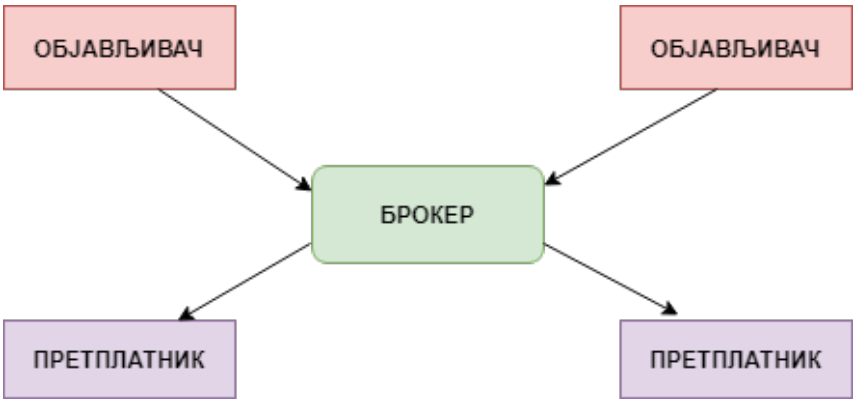
Слика 2.1: *Захтев-Одговор* модел

2.2 Модел објава-претплата

Код модела објава-претплата (енг. *publish-subscribe*) постоје 2 врсте клијената :

- објављивач (енг. *publisher*) - снабдевач информација на одређену тему,
- претплатник (енг. *subscriber*) - онај који добија информације на тему на коју се претплатио.

До размене информација долази преко треће компоненте - брокера (сервера, посредника). Наиме, објављивач шаље поруку брокеру са идентификатором који означава њену тему или област. Брокер затим прослеђује добијену поруку свим претплатницима на дату тему. На тај начин клијенти не морају да знају једни за друге и не морају бити активни у исто време. Клијент не мора да захтева информације, него их брокер гура ка клијенту у време када има информације доступне за њега. Захваљујући овим карактеристима се постиже асинхрона комуникација јер ниједна страна није блокирана у току размене информација. Брокер игра улогу пивота, који филтрира и прослеђује поруке. Неки брокери пружају могућност повезивања са другим брокерима путем такозваних мостова, што омогућава већи број повезаних клијената јер клијенти повезани са једним сервером могу добијати поруке објављене на други сервер.
Овај модел комуникације представља комбинацију један-на-нула, један-на-један и један-на-више дистрибуције података. О моделу објава-претплата се може више наћи у [3].



Слика 2.2: *Објава-Претплата* модел

Глава 3

Протокол *MQTT*

Протокол *MQTT* (слика 3.1 [1]) је клијент-сервер протокол за размену порука чији се рад заснива на примени модела објава-претплата. Отвореног је стандарда, једноставан и дизајниран да буде што лакши за имплементацију. Ове карактеристике га чине идеалним за употребу у многим ситуацијама, укључујући ограничена окружења попут M2M (енг. *Machine to machine*) комуникације и *IoT* (енг. *Internet of things*) где су мале количине кôда и мрежна пропусност добитак [2].

Слика 3.1: *MQTT* лого

3.1 Историја протокола *MQTT*

Протокол *MQTT* су, 1999. године, развила 2 инжењера : Енди Стенфорд-Кларк из *IBM*-а и Арлем Нипер из *Eurotech*-а. Протокол *MQTT* је настао из њихове потребе за новим протоколом који је морао да подржи неколико захтева међу којима су једноставна имплементација, континуирано одржавање сесије, квалитет услуге

[1] Слика је преузета са сајта http://mqtt.org/

при испоручивању порука, итд.
Наредних 10 година *IBM* га је користио интерно, до 2010. године, када су објавили *MQTT v3.1* отворену верзију протокола. Године 2014. протокол *MQTT v3.1.1* је постао званично одобрен *OASIS* стандард, а затим 2016. године постао је и званични *ISO* стандард под називом *ISO/IEC 20922:2016*. Априла 2019. године *OASIS* је објавио званичну верзију *MQTT v5.0*.
Данас, захваљујући својим карактеристикама, протокол *MQTT* представља један од најпопуларнијих протокола у *IoT* индустрији.

3.2 Структура и начин функционисања верзије 3.1.1 протокола *MQTT*

MQTT је апликативни протокол који ради на врху *TCP/IP* модела, што значи да размена података између клијента и сервера може започети након што клијент успостави са сервером *TCP/IP* везу на одређеном мрежном комуникационом порту (порт 1883 је резервисан за стандарну конекцију, а 8883 за конекцију преко *SSL*-а). Као што је напоменуто раније, у моделу објава-претплата се разликују клијенти који примају информације од оних који их објављују и таква је ситуација и у протоколу *MQTT*, али је важно напоменути да један клијент може обављати обе функције. Комуникација се врши искључиво између клијената и сервера. Клијенти међусобно не знају за постојање и на тај начин је између њих постигнута временска и просторна независност будући да не морају бити активни у исто време, а и не зависе од података о физичким локацијама других клијената (попут *IP* адресе и порта).
Сервер чува податке о клијентима и њиховим сесијама. Подаци о једном клијенту укључују информације о свим претплатама клијента, све поруке од интереса [2] које му нису испоручене (јер није био доступан на мрежи), поруке чија размена (енг. *handshake*) није завршена, као и тестамент поруке које се објављују у случају да дође до неочекиваног прекида везе са клијентом (у наставку детаљније појашњење). Подаци се чувају по клијентским идентификаторима, који морају бити јединствени по клијенту. Наиме, када клијент успоставља везу са сервером, шаље податак који указује на то да ли клијент жели да сесија буде нова или да се настави на стару (ако таква постоји) - о овоме више у делу 3.2.3.1.

[2] Поруке од интереса представљају поруке на чије теме се клијент претплатио.

3.2.1 Теме у протоколу *MQTT*

У протоколу *MQTT* клијент објављује поруке по темама, које се могу посматрати као области интересовања. Сервер филтрира примљене поруке по темама и прослеђује их заинтересованим клијентима (онима који су се претплатили на дате теме).
Тема је представљена као *UTF-8* стринг дужине веће од један, где *UTF-8* представља начин кодирања карактера. Свака тема се састоји од једног или више нивоа који су раздвојени помоћу **карактера слеш** /.
Пример теме са три нивоа : *Srbija/Beograd/Temperatura* где *Srbija* представља први ниво, *Beograd* други ниво, а *Temperatura* трећи ниво.
При претплати на тему, постоје и специјални знакови који омогућавају претплату на више тема одједном.

- Употреба **плус карактера (+)** замењује један ниво у хијерархијској организацији теме, тј. замењује конкретан стринг у нивоу. Ако би се клијент претплатио на тему *Srbija/+/Temperatura* тада би добијао поруке на све теме тог облика, на пример *Srbija/Jagodina/Temperatura*, *Srbija/Beograd/Temperatura*, а не би добијао поруке на теме попут *Srbija/Temperatura* или *Srbija/Beograd*.

- Употреба **тараба карактера (#)** замењује више нивоа. У конкретном примеру претплата на тему *Srbija/#* значила би претплату на све теме којима је први ниво једнак *Srbija*, као што су *Srbija/Subotica*, *Srbija*, *Srbija/Valjevo/Voda* итд. Карактер # мора бити последњи у хијерархијској организацији. Уколико клијент жели да се претплати на све теме потребно је да тема садржи само карактер #.

Употреба специјалних знакова + и # није дозвољена при објављивању тема и ван ознака за хијерархију нивоа, на пример *Srbija/Beograd+* или *Beograd/#Ulica* нису валидне теме. Потребно је напоменути да се у темама разликују велика и мала слова, тако да теме *Temperatura* и *temperatura* нису једнаке.
Теме чији је први карактер једнак карактеру **$** се посматрају као специјалне теме, и клијентима није дозвољено објављивање таквих тема, али могућа је претплата. Опште је прихваћено да поруке чије теме почињу са називом **$SYS** садрже интерне информације о серверу. Претплата на све поруке (употребом само карактера #) не обухвата претплате на теме чији је први карактер **$**.

3.2.2 Квалитет услуге

Када је реч о размени порука, тада се мора говорити о квалитету услуге (енг. *Quality of service*, у наставку *QoS*) који представља једну од најбитнијих карактеристика протокола *MQTT*. Када клијент објављује поруку на одређену тему, тада уз поруку као један од података шаље и њен ниво *QoS*-а који представља степен сигурности по коме та порука треба да се испоручи. Постоје 3 нивоа *QoS*-а.

- *QoS* 0 (највише једном) - не постоји гаранција о испоруци. Пошаљилац шаље једном и не добија потврду о испоруци.
- *QoS* 1 (најмање једном) - порука се испоручује најмање једном. Пошаљилац поруку шаље изнова све док не добије потврду о пристизању поруке.
- *QoS* 2 (тачно једном) - порука се испоручује тачно једном и за то је заслужан механизам четвороструког руковања (енг. *handshake*).

На исти начин када се клијент претплати на неку од тема, он наведе ниво *QoS*-а по коме жели да прими поруку на ту тему. Међутим може доћи до ситуације да ниво *QoS*-а који је дефинисан у објављеној поруци буде различит од оног наведеног у претплати. Када клијент објави тему уз одређени ниво *QoS*-а та тема ће се на том нивоу испоручити серверу и чуваће се као податак о тој поруци. Међутим, ако клијент који се претплатио на тему постави нижи ниво *QoS*-а, сервер ће применити нижи ниво при достављању поруке. О овоме више у секцији 3.2.3.2.

3.2.3 Врсте порука у протоколу *MQTT*

Одржавање везе, објављивање порука на одређене теме, претплате на теме, итд. су процеси који се одвијају помоћу размене различитих врста порука. Постоји 14 врста порука (табела 3.1) и њиховом применом, на предефинисан начин, се омогућује функционисање протокола. Поруке у протоколу *MQTT* су познате и под називом „контролни пакети“ [2] где је потребно нагласити да се термин *пакет* не односи на пакете из транспортног слоја[3].

[3]Податке добијене из слоја апликација TCP дели на пакете одређене величине које прослеђује међумрежном слоју.[1]

Једна од позитивних карактеристика протокола *MQTT* се огледа у његовој ефикасности која не оптерећује проток. Управо та ефикасност потиче од организације порука чија је величина минимална и прилагођена ограниченим окружењима. Поруке се састоје од три целине, једне обавезне и две опционе. Целине су увек наведене у истом редоследу:

- **фиксирано заглавље** представља обавезну целину која се налази у свакој поруци и која садржи информацију о типу поруке и величини преосталог дела поруке, величина овог заглавља је 2 бајта;
- **варијабилно заглавље** је целина коју садрже неки типови поруке, његов садржај варира од типа поруке;
- **садржај** (енг. *payload*) - представља целину коју садрже неке поруке и у њој се налазе информације о садржају објављених порука, тема за претплату, тестамент порука, итд.

Више о структури порука се може пронаћи у поглављу 2 из [2].

Табела 3.1: Врсте порука у протоколу *MQTT*

Назив	**Опис**
CONNECT	Захтев клијента за повезивање са сервером
CONNACK	Серверска потврда о повезивању
PUBLISH	Објављивање поруке
PUBACK	Потврда о примању поруке *PUBLISH*
PUBREC	Потврда о примању поруке *PUBLISH*
PUBREL	Потврда о примању поруке *PUBREC*
PUBCOMP	Потврда о примању поруке *PUBREL*
SUBSCRIBE	Захтев клијента за претплату на тему
SUBACK	Серверска потврда о претплати
UNSUBSCRIBE	Клијентски захтев за укидање претплате
UNSUBACK	Серверска потврда о укидању претплате
PINGREQ	Клијентско одржавање сесије
PINGRESP	Серверски одговор за одржавање сесије
DISCONNECT	Клијентски захтев за прекид везе

3.2.3.1 Успостављање и прекид везе

Да би започео комуникацију, клијент шаље серверу поруку *CONNECT*. Исправна *CONNECT* порука садржи податке о:

- клијентском идентификатору (једини обавезан податак) - служи за идентификовање клијената који су повезани са сервером, мора бити јединствен;
- индикатору за чисту сесију (енг. *cleanSession*) који указује на то да ли сесија треба да се настави на постојећу (ако таква постоји). Ово је битно јер сервер може чувати податке о ранијим претплатама клијента, као и порукама од интереса које треба да му стигну. Овај податак указује и да ли сервер треба да чува новонастале претплате;
- корисничком имену и лозинци који служе као додатни корак у проверавању идентитета клијента. Битно је напоменути да се подаци не шаљу шифровани, него у виду чистог текста;
- тестамент поруци - порука на одређену тему коју клијент жели да објави уколико до раскида везе дође на неочекиван начин (уколико дође до неке грешке и клијент не пошаље захтев за раскид везе). Подаци обухватају назив теме, садржај поруке, ниво *QoS*-а и индикатор *retain* који ће бити накнадно објашњен;
- индикатору *keepAlive* који се користи за одржавање трајања сесије. Означава максимално време које може да прође, а да не дође до размене порука.

CONNACK представља серверски одговор на захтев за повезивање. Садржи два податка :

- флег о присутној сесији (*sessionPresent*),
- повратни кôд.

Вредност флега *sessionPresent* зависи од података из поруке *CONNECT*, као и доступних информација на серверу. Што се тиче повратног статуса, за сада постоји 6 врста (табела 3.2) док је преосталих 250 резервисано за будућу употребу (будући да је статус представљен као 8-битни број).

Када клијент жели да прекине везу, потребно је да пошаље поруку *DISCONNECT* и тада сервер по примању наведене поруке брише податке о тестамент поруци и прекида везу са клијентом.

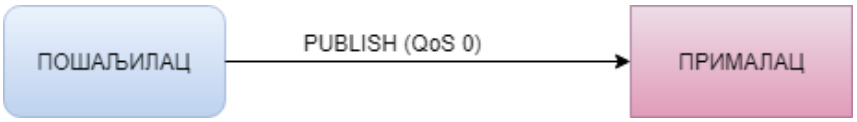
Табела 3.2: Повратни статус у поруци *CONNACK*

Вредност	**Опис**
0	Повезивање прихваћено
1	Повезивање одбијено, неприхватљива верзија протокола
2	Повезивање одбијено, одбијен идентификатор
3	Повезивање одбијено, сервер недоступан
4	Повезивање одбијено, лоше корисничко име или лозинка
5	Повезивање одбијено, клијент нема ауторизацију

3.2.3.2 Објављивање поруке

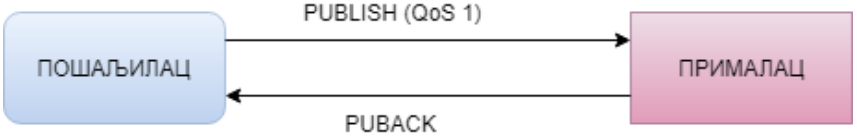
Када клијент жели да објави или сервер жели да проследи поруку на одређену тему тада користе *PUBLISH* тип поруке. Порука *PUBLISH* садржи податке о теми и садржају поруке, индикаторе *dup* и *retain*, и ниво квалитета услуге по коме је потребно доставити поруку. Садржаји порука у протоколу *MQTT* су бинарни, тако да садржај може бити произвољног формата и садржаја.
Када клијент пошаље *retain = true* индикатор при објављивању порука, он указује на то да жели да се та порука чува као последња валидна порука на дату тему, што значи да ће сваки нови претплатник на наведену тему добити поруку коју је сервер сачувао као последњу валидну поруку за ту тему. Ако је сервер већ имао сачувану поруку на задату тему, тада ће он обрисати постојећу и додати нову.
Као што је наведено раније, постоје 3 нивоа *QoS*-а, и у зависности од наведеног постиже се одговарајући степен сигурности при испоручивању поруке.

- Када је *QoS* = 0, тада објављивач поруке (клијент који објављује или сервер који прослеђује) само пошаље поруку *PUBLISH* и не очекује повратну информацију о њеном пристизању (слика 3.2).



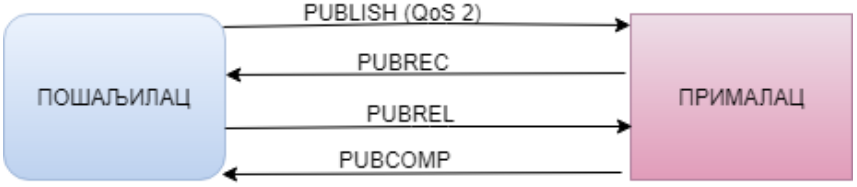
Слика 3.2: Објављивање поруке са *QoS* 0 нивоом

- Када је *QoS* = 1 тада објављивач поруке *PUBLISH* очекује да ће се порука испоручити бар једном, те ће, уколико у догледно време не добије одговор о његовој испоруци, изнова слати ту поруку са разликом у томе што ће индикатор *dup* бити постављен на 1 (односно биће *true*) јер управо он указује да је порука поново послата. Објављивач ће слати поруку изнова све док не добије потврду или док не добије грешку од *TCP* слоја. Одговор на *PUBLISH* представља *PUBACK* (слика 3.3).



Слика 3.3: Објављивање поруке са *QoS* 1 нивоом

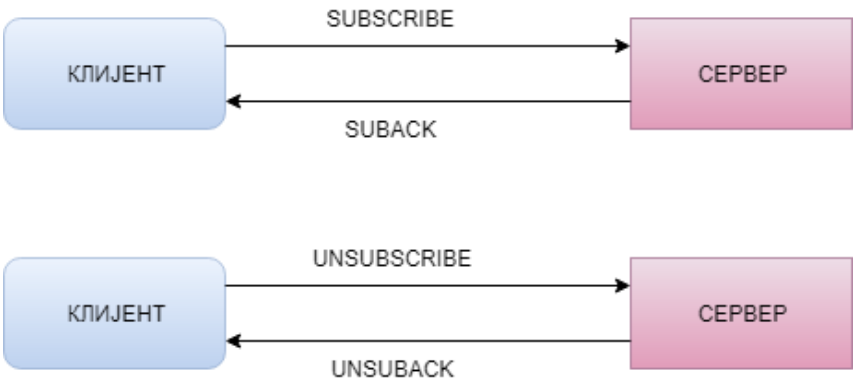
- Када је *QoS* = 2 тада објављивач поруке *PUBLISH* очекује да се да се она испоручи тачно једном, и у томе му помаже механизам четвороструког руковања који се постиже помоћу порука *PUBREC, PUBREL* и *PUBCOMP*. Наиме, када објављивач пошаље поруку *PUBLISH*, он као одговор очекује поруку *PUBREC*. Уколико у догледно време не добије одговор, објављивач ће изнова слати *PUBLISH* са постављеним *dup* индикатором. Када прими одговор у виду поруке *PUBREC*, објављивач шаље поруку *PUBREL* и очекује одговор у виду поруке *PUBCOMP*.



Слика 3.4: Објављивање поруке са *QoS* 2 нивоом

3.2.3.3 Претплата на тему

Да би примио поруке на одређену тему, клијент се најпре мора претплатити на ту тему, тј. мора послати поруку *SUBSCRIBE*. У наведеној поруци је потребно да се налази листа тема са њиховим максималним *QoS* нивоима (нивои са којима му сервер може послати поруке са задатом темом). Уколико је клијент већ имао претплате на неку од тема на које жели да се претплати, сервер ће прегазити постојеће. Сервер као одговор на *SUBSCRIBE* шаље поруку *SUBACK* (слика 3.5) која садржи повратни кôд за сваки од парова тема/*QoS* ниво. Може се и напоменути да сервер може проследити поруку клијенту на тему коју се управо претплатио, чак и пре слања поруке *SUBACK*.

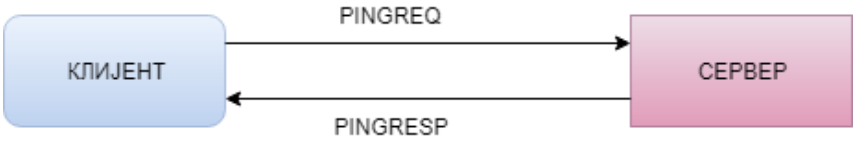


Слика 3.5: Претплаћивање и одјава теме

Уколико клијент жели да прекине претплату на одређену тему, потребно је да пошаље поруку *UNSUBSCRIBE* која садржи податке о листи тема са којих клијент жели да се одјави. Уколико је при примању поруке *UNSUBSCRIBE* сервер био у процесу слања порука на неку од тема са којих клијент жели да се одјави, сервер мора завршити слање наведених порука, али треба и да спречи слање нових. Сервер сваку тему проверава карактер по карактер, тако да ће уклонити само оне које се тачно поклапају (без обзира на употребу специјалних карактера). Клијент добија одговор од сервера у виду поруке *UNSUBACK* (слика 3.5). Након добијеног одговора, клијент може да рачуна да су претплате на наведене теме обрисане.

3.2.3.4 Одржавање сесије

Након повезивања клијента и сервера, може се десити да одређени временски период не дође до размена порука, у таквој ситуацији се поставља питање како обе стране знају да ли је друга страна и даље активна и да ли је конекција још увек исправна?
Механизам који је одговоран за одржавање истрајне сесије се одвија захваљујући порукама *PINGREQ* и *PINGRESP* (слика 3.6), као и *Keep Alive* периоду који је прослеђен приликом слања поруке *CONNECT* (секција 3.2.3.1). Ако је *Keep Alive* различит од нуле онда тај број представља интервал за највећи временски период у секундама који може да прође, а да не дође до размена порука. Клијент, уколико је активан и уколико није дошло до размена порука, је дужан да након истека периода пошаље поруку *PINGREQ*. Сервер (уколико је активан) одговара са поруком *PINGRESP* и на тај начин даје клијенту до знања да је и он активан. Уколико



Слика 3.6: Одржавање сесије

клијент не пошаље *PINGREQ* за временски период *Keep Alive* * 1.5, сервер закључује да клијент више није активан и наставља са прекидањем везе са клијентом. И слично, уколико сервер не одговори одређени временски период, клијент доноси закључак да сервер више није активан.
Када се наведе да је *Keep Alive* = 0, онда је механизам одржавања сесије искључен и сервер није дужан да прекине везу са клијентом услед неактивности.
Будући да је представљен ако 16-битни број, максимална вредност за *Keep Alive* период износи 65535, односно 18 сати 12 минута и 15 секунди.

3.3 Сигурност у протоколу *MQTT*

Потребно је размотрити различите аспекте сигурности, међу којима су :

- провера идентитета клијента,

- ауторизација,
- безбедносни механизми.

Провера идентитета и ауторизација се могу вршити помоћу података које клијент обезбеђује приликом повезивања. Као што је раније напоменуто у делу 3.2.3.1, приликом повезивања клијент шаље серверу податке попут клијентског идентификатора и корисничког имена и лозинке на основу којих сервер врши идентификацију. Већина сервера (пример *Mosquitto, HiveMQ*) пружа додатне мехнизме за ауторизацију где се, у зависности од корисничких података, клијенту омогућава или одбија приступ одређеним темама - како за објављивање, тако и за претплате (у наставку ће бити објашњен механизам за *Mosquitto*). Међутим, корисничко име и лозинка се шаљу у виду чистог текста, што представља опасност за сигурност. Протокол *MQTT* је транспортни протокол који не садржи безбедносне механизме, одговорност у безбедносној имплементацији је у рукама онога који развија софтвер. Два најчешћа приступа при решавању овог проблема су сигурно повезивање и шифровање података.

- Сигурно повезивање се обезбеђује помоћу *TLS*-а (енг. *Transport Layer Security*) који омогућује слој испод протокола *MQTT*, тачније *TCP/IP*, више о овоме се може наћи у [1]. Овај приступ омогућује заштиту и корисничких података и садржаја објављене поруке.
- Шифровање садржаја објављене поруке. Овај приступ би требало омогућити са клијентске стране будући да сервер преноси поруке у облику у коме стигну до њега. Што значи да би клијент који објављује поруке исте треба да шифрује, а клијент који их прима би требао да их дешифрује. Овај приступ не обезбеђује заштиту лозинке при успостављању везе.

3.4 Протоколи *MQTT* и *HTTP*

Протокол *Hypertext Transfer Protocol* (*HTTP*) представља један од најпознатијих протокола чија примена лежи у размени информација на вебу. Протокол *HTTP* се такође налази на врху модела *TCP/IP* и врло често се врше поређења између ова два протокола у погледу примене у *IoT* средини. Постоји више аспеката по којима се ова два протокола могу поредити, али према [3] издваја се неколико.

Дизајн

- У протоколу *MQTT* садржај и формат порука нису битни будући да се садржај преноси као низ бајтова.
- Протокол *HTTP* подржава *MIME* стандарде за дефинисање типа садржаја, ограниченим уређајима углавном није потребна ова функционалност.

Механизам размене порука

- Протокол *MQTT* користи модел објава-претплата који омогућава да клијенти међусобно не знају за постојање ни да буду активни у исто време, њихова једина комуникација је са сервером.
- Протокол *HTTP* користи модел захтев-одговор за размену порука и клијент у овом случају мора да зна за све учеснике са којима жели да оствари комуникацију.

Сложеност протокола

- Протокол *MQTT* има кратку спецификацију, и основне поруке о којима развијаци треба да воде рачуна су **CONNECT, PUBLISH, SUBSCRIBE, UNSUBSCRIBE** и **DISCONNECT**.
- Протокол *HTTP* има сложенију спецификацију, користи много повратних кôдова и метода.

Величина порука

- Будући да је направљен специјално за ограничене уређаје, *MQTT* поруке су осмишљене тако да буду минималне и ефикасне. Заглавље *MQTT* поруке је величине 2 бајта.
- Поруке у протоколу *HTTP* су представљене у текстуалном формату што као последицу даје обимнија заглавља и садржај. Формат *HTTP* поруке је читљив људима и на тај начин омогућава лакше уочавање проблема, али у околинама са ограниченим уређајима та особина је захтевна и сувишна.

Квалитет услуге

- Протокол *MQTT* има дефинисана 3 нивоа квалитета услуге и програмери не морају да имплементирају додатне функционалности да би осигурали испоручивање порука.
- Протокол *HTTP* нема дефинисан квалитет услуге, све поруке се испоручују са истом нивоом квалитета, тако да су потребне додатне имплементације уколико је потребна већа разноликост у квалитету размена порука.

Дистрибуција података

- Протокол *MQTT* омогућује размену података један-на-нула, један-на-један и један-на-више.
- Протокол *HTTP* омогућује само један-на-један размену података.

Глава 4

Алати и окружења

4.1 *Eclipse Paho* и *M2Mqtt*

Пројекат *Eclipse Paho* представља скуп клијентских имплементација протокола *MQTT* на различитим платформама, укључујући *C#, C, Python,* итд. Пројекат је настао под покровитељством *Eclipse Foundation* (слика 4.1[1]) која представља непрофитабилну организацију, основану од стране *IBM*-а 2001. године, а чији су сврха и циљ ширење и побољшање заједнице отворених стандарда (енг. *open source*).

Слика 4.1: *Eclipse Foundation* лого

Апликација развијена за потребе рада имплементира једну од *Eclipse Paho* библиотека, направљених за *.NET* окружење. У питању је библиотека *M2Mqtt* која *.NET* програмерима пружа интерфејс за пуну примену *MQTT* протокола без потребе за

[1]Слика је преузета са Википедије

знањем о начину његове имплементације. Међу њене карактеристике спадају неблокирајући (асинхрони) *API*-ји, имплементација 3.1.1 верзије протокола *MQTT*, подршка за стандардну *TCP* конекцију, као и за *SSL/TLS* сигурну конекцију, итд.

Пример кôда 4.1: Делови класе *MqttClient* из библиотеке *M2MQtt*

```
public class MqttClient
    {
        ...

        public byte Connect(string clientId, string username, string
                    password, bool willRetain, byte willQosLevel, bool
                    willFlag, string willTopic, string willMessage, bool
                    cleanSession, ushort keepAlivePeriod);
        public byte Connect(string clientId, string username, string
                    password, bool cleanSession, ushort keepAlivePeriod);
        public byte Connect(string clientId, string username, string
                     password);
        public byte Connect(string clientId);
        public void Disconnect();
        public ushort Publish(string topic, byte[] message);
        public ushort Publish(string topic, byte[] message, byte qosLevel,
                      bool retain);
        public ushort Subscribe(string[] topics, byte[] qosLevels);
        public ushort Unsubscribe(string[] topics);

        ...
    }
```

Као што се може видети у примеру кôда 4.1, обрађене су операције везане за комуникацију са сервером. Ако се за пример узме метода *Connect*, може се видети да постоје 3 различита потписа која се могу применити у зависности од жељеног резултата. Аналогно важи и за остале методе.
Детаљи наведене библиотеке се могу наћи на следећем урл-у: *m2mqtt.wordpress.com*, делови примене се могу видети и у поглављу 5. Библиотека *M2Mqtt* је доступна и на *NuGet*-у који представља менаџер пакета помоћу кога програмери могу да праве, објављују и користе пакете и библиотеке који се користе за програмирање у *.NET* оквиру. Више о *NuGet*-у се може наћи у [5].

4.2 *Eclipse Mosquitto* сервер

Пројекат *Eclipse Mosquitto* (слика 4.2 [2]) пружа имплементацију *MQTT* брокера под називом *Mosquitto*. *Eclipse Mosquitto* је као и *Eclipse Paho* део организације *Eclipse Foundation*.

Слика 4.2: *Mosquitto* лого

Mosquitto подржава верзије 3.1 и 3.1.1 протокола *MQTT* (у време писања рада подржане су неке опције верзије 5, али модел није био употпуњен), ефикасан је и погодан је за употребу на свим уређајима - од слабијих рачунара са малим напајањем до моћнијих сервера. Поред брокера, *Mosquitto* пројекат обезбеђује и библиотеке за имплементацију клијената у програмском језику *C*, као и конзолне апликације *mosquitto_pub* и *mosquitto_sub* које представљају имплементације клијената који се могу конектовати на *Mosquitto* сервер и објављивати и претплаћивати се на теме.
Mosqitto је могуће преузети са званичног сајта (*mosquitto.org*). Након преузимања, сервер је потребно инсталирати и покренути. Да би се *Mosqitto* покренуо, потребно је покренути *Command Prompt*, позиционирати се у оквиру инсталационог директоријума и покренути наредбу **mosquitto**. Опционо, могуће је тестирати рад сервера помоћу конзолних апликација *mosquitto_sub* и *mosqitto_pub*.
У оквиру инсталационог директоријума, постоји и одговарајући конфигурациона датотека (*mosquitto.config*) помоћу које је могуће дефинисати различита подешавања за *Mosqutto*. Подешавања обухватају информације попут: дефинисања више портова, путање до сертификата за сигурну конекцију, конфигурисање мостова итд. У конфигурационој датотеци се могу навести и путање до датотека у којима су предефинисани корисници (ту се налазе подаци о корисничким именима и лозинкама) и листе за ауторизацију (енг. *Access control list - ACL*) у којој је могуће поставити ограничења везана за теме на које се клијенти могу претплатити и на

[2] Слика је преузета са https://mosquitto.org/

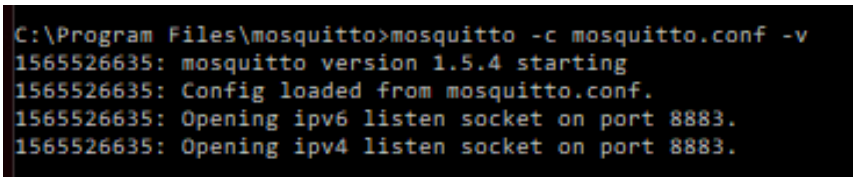
које могу објавити. Да би наведена конфигурациона датотека могла да се користи, приликом сваке измене података у њој или у датотекама на које реферише, потребно је поновно покретање *Mosquitto* сервера, што може бити незгодно у ситуацијама када се корисници додају динамички. У таквој ситуацији, постоје занимљиви додаци (енг. *plugins*) за *Mosquitto* сервер, који управо омогућавају рад са динамичким корисницима, и пример једног таквог додатка је *mosquitto-auth-plug* (који се може пронаћи на урл-у https://github.com/jpmens/mosquitto-auth-plug) помоћу кога је могуће користити базе података за складиштење одговарајућих информација о корисницима, те није неопходно поновно покретање сервера у случају промене. Мана овог додатка је што није прилагођен за *Windows* оперативне системе већ само за *Linux*.
Као што је раније напоменуто протокол *MQTT* није сигуран, тачније потребна су додатна подешавања да би се размена порука обезбедила. *Mosquitto* пружа подршку за сигурну *TLS/SSL* конекцију и да би се таква конекција остварила потребно је у *mosquitto.config* датотеци обезбедити путање до одговарајућих сертификата, подешавање порта на 8883 као и (опционо) навести верзију *TLS*-а (слика 4.3).

```
port 8883
cafile C:\Program Files\mosquitto\certs\ca.crt
keyfile C:\Program Files\mosquitto\certs\server.key
certfile C:\Program Files\mosquitto\certs\server.crt
tls_version tlsv1.2
```

Слика 4.3: Подешавања конфигурационе датотеке за *Mosquitto* ради обезбеђивања сигурне конекције

За потребе рада коришћени су самопотписани сертификати (енг. *self-signed*) који су обезбеђени помоћу *OpenSSL* алата. Сертификат се у стварном животу може посматрати као лична карта која пружа за проверу идентитета њеном власнику и личну карту мора издати проверена странка (енг. *Certificate Authority- CA*) у овом случају орган власти који гарантује за идентитет власника личне карте. Када се креира самопотписани сертификат то је као да неко сам себи издаје личну карту те је потребно да се на неки начин региструје као адекватна и проверена особа за ту активност.

Да би се му се указало да треба да користи конфигурациону датотеку, *Mosquitto* је потребно покренути као на слици 4.4 где опција *-v* обезбеђује логовање активности.



Слика 4.4: Покретање *Mosquitto* сервера

4.3 *Xamarin.Forms*

Нативна апликација представља софтвер развијен за одређену мобилну платформу (*Android*, *iOS*), која међу своје функционалности ставља додатке доступне на уређају (као што су камера, сензори, микрофон). Овакав приступ захтева познавање различитих програмских језика за различите платформе и познавање специфичности различитих платформи јер већина платформи подржава само апликације развијене за њу. За развој *Android* нативне мобилне апликације потребно је познавање програмског језика *Java*, док је за *iOS* у питању *Swift*. То значи да развој исте апликације за различите платформе може представљати веома незгодан и мукотрпан посао. У таквој ситуацији решење долази у виду међуплатформских (енг. *cross platform*) оквира за израду мобилних апликација који омогућавају да једном развијена мобилна апликација може да се покрене на различитим платформама. Један од најпопуларнијих међуплатформских оквира представља *Xamarin*.

Xamarin (слика 4.5 [1]) представља скуп алат који омогућавају развој вишеплатформских мобилних апликација помоћу *C#*-а.

Слика 4.5: *Xamarin* лого

Xamarin је компанија основана 2011. године чији су оснивачи радили на *Mono* платформи која представља отворену (енг. *open-source*) имплементацију *.NET* окружења која може да се покреће на различитим оперативним системима. Ово је битно из разлога што су *Xamarin.Android*, *Xamarin.iOS* и *Xamarin.Forms* развијени помоћу *Mono* платформе.

Xamarin компанија је 2016. постала део Мајкрософт компаније и за развој *Xamarin*

мобилних апликација користи се *Visual Studio* развојно окружење. *Xamarin.Forms* је изграђен помоћу *Xamarin.Android* и *Xamarin.iOS* и садржи јединствен *API* за рад са више платформи. *Xamarin.Forms* представља скуп алата који омогућава развој заједничког корисничког интерфејса за више платформи. Корисничи интерфејс у *Xamarin.Forms* апликацијама се дефинише користећи *XAML* (енг. *Extensible Application Markup Language*), а то је језик заснован на *xml*-у.

Будући да је апликација израђена за потребе рада развијена помоћу *Xamarin.Forms*, делови имплементације се могу видети у поглављу 5. Више о *Xamarin.Forms* се може наћи у [4].

Глава 5

Апликација

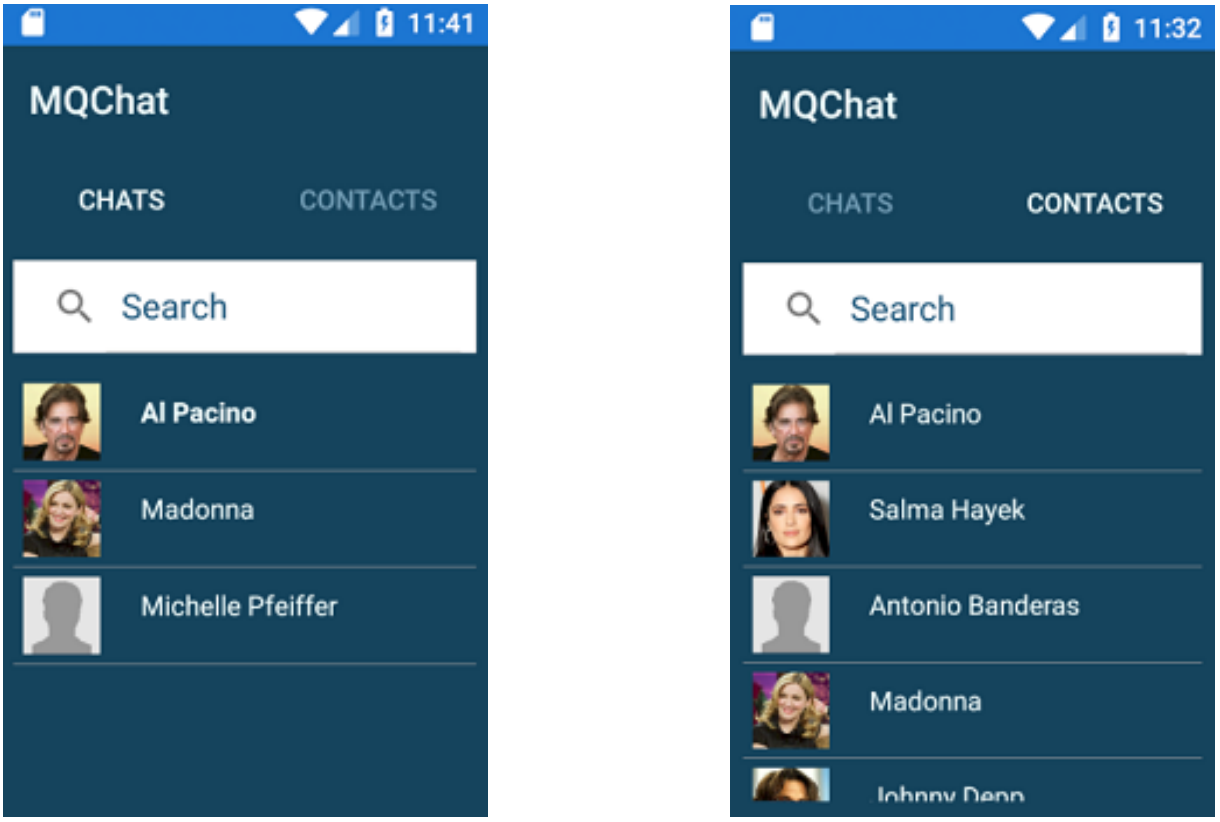
Протокол *MQTT* има највећу примену у *IoT* области, међутим због својих карактеристика протокол *MQTT* је погодан за примену у различитим сферама мрежне размене података, па тако може имати примену и у друштвеним мрежама. Познато је да *Facebook Messenger* ради помоћу протокола *MQTT*, међутим није познато у којој мери и на који начин.
MQChat представља апликацију израђену за потребе рада као један од предлога његове могуће примене у својству размена порука на друштвеним мрежама.

5.1 Кориснички интерфејс и упутство за употребу

Приликом покретања апликације, уколико корисник није доделио права приступа контактима, биће му приказан упитни дијалог у коме се корисник пита да ли жели да додели апликацији право за приступ контактима.

- Уколико одабере одричан одговор апликација неће наставити са радом.
- Уколико корисник одабере потврдан одговор биће му приказана страница за пријављивање и након уноса одговарајућих података, корисник се преусмерава на почетну страницу апликације (слика 5.1).

На почетној страници инцијално је учитана страница *CHATS* где су приказани сви контакти са којима је корисник остварио комуникацију путем апликације *MQChat*



Слика 5.1: Странице *CHATS* и *CONTACTS*

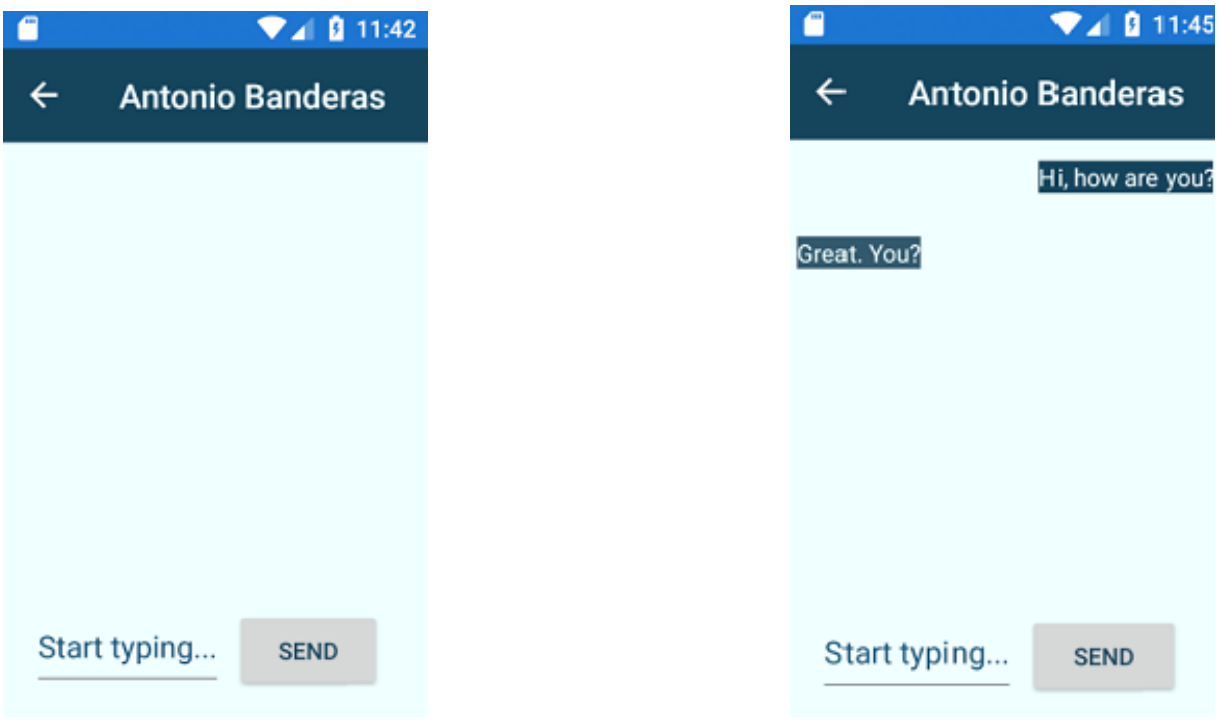
(уколико таквих има). Контакти са којима корисник има непрочитане поруке су назначени подебљаним словима. Корисник може да изврши претрагу путем поља за претрагу.

Одабиром поља *CONTACTS*, кориснику се приказује страница са свим контактима на којој је такође могуће вршити претрагу. На овој страници су приказани само контакти и нису назначени они са којима корисник има непрочитане поруке.

Одабиром контакта преко странице *CHATS* или *CONTACTS*, приказује се страница за размену порука са одабраним контактом (слика 5.2). Поруке су приказане у хронолошком редоследу, где се при дну странице налазе најновије. Корисник може да пошаље поруку уносом текста у поље за текст и одабиром дугмета *Send*.

5.2 Структура пројекта

MQChat представља апликацију која се састоји из два пројекта: *MqttChatClient* и *MqttChatClient.Android*, где је у првом дефинисан кориснички интерфејс и дељени кôд (заједнички за све платформе, у овом случају само *Android*), док се помоћу



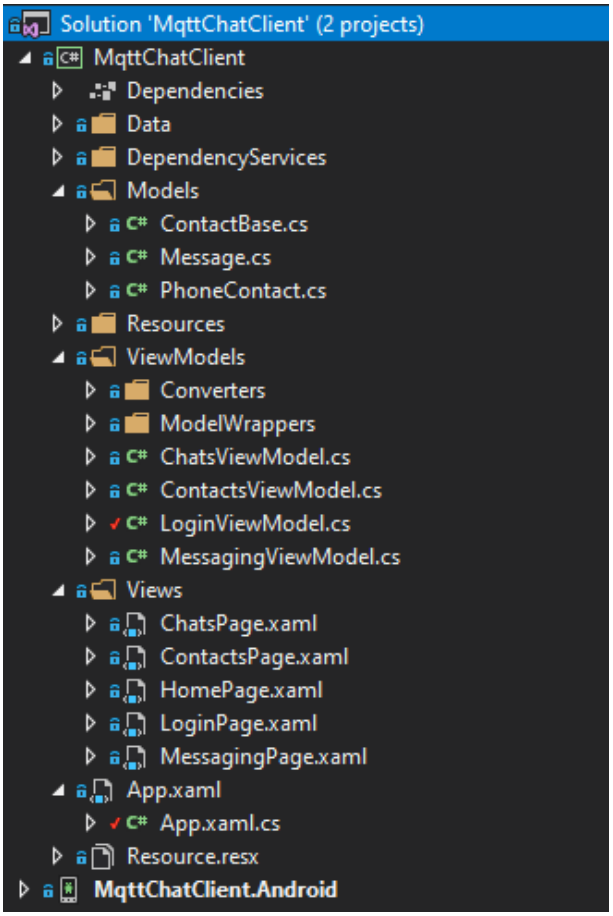
Слика 5.2: Размена порука

другог добијају информације везане за *Android* платформу. Целокупан изворни кôд апликације доступан је на адреси : https://github.com/alehandra/MqttChatClient. Апликација *MQChat* је реализована помоћу архитектуалног шаблона **Модел-Поглед-Поглед-Модел** (МППМ) који олакшава раздвајање бизнис логике од корисничког интерфејса. Укратко, МППМ се састоји из следећих компоненти:

- *Модел* - подаци из пословне логике,
- *Поглед* - дефинисан кориснички интерфејс,
- *Поглед-Модел* - омогућује приказивање података из Модела на једноставан и интуитиван начин.

На слици 5.3 приказана је хијерархијска организација пројекта *MqttChatClient* и ту се такође могу видети, издвојени по директоријумима, елементи МППМ шаблона.

Модели из пословне логике су представљени путем класа *Message.cs* и *PhoneContact.cs*. Помоћу објеката класе *PhoneContact* су представљени контакти који садрже својства попут броја телефона, имена и слике контакта (пример кôда 5.1).



Слика 5.3: Структурна организација *MQChat* апликације

Пример кôда 5.1: Класа *PhoneContact*

```
public string FirstName { get; set; }
public string LastName { get; set; }
public string PhoneNumber { get; set; }
public ImageSource ImageSource { get; set; }

public string Name { get => $"{FirstName} {LastName}"; }
```

Помоћу класе *Message.cs* су описане поруке које се размењују између корисника путем апликације *MQChat*. Класа садржи својства попут садржаја поруке, времена креирања, примаоца и пошаљиоца итд.

Почетна страница апликације (*HomePage.xaml* са слике 5.3) представља *TabbedPage* са које је могуће приступити страницама *ChatsPage.xaml* и *ContactsPage.xaml*. Наведене странице представљају Погледе из МППМ шаблона и имају одговарајуће

Поглед-Моделе *ChatsViewModel.cs* и *ContactsViewModel.cs*. На обе странице је приказана компонента *ListView* која се везује са листама контаката дефинисаним у одговарајућим Поглед-Моделима.
Један од битних појмова у *Xamarin.Forms* представља везивање података (енг. *Data Binding*) који омогућава да подаци из два објекта буду повезани тако да промене у једном изазову промене у другом. Тако посматрајући, подаци из једног објекта ће представљати извор, а подаци из другог циљ. Примера ради, текст ознаке (енг. *Label*) у неком Погледу се може везати за својство (енг. *Property*) неке класе и уколико би дошло до промене вредности поменутог својства, промене би биле видљиве и на ознаци. Овде се може споменути интерфејс *IValueConverter* који омогућује везивање података различитих типова (на пример пребројиви тип података се може везати за стринг).
Пример за везивање података се може наћи у свим Погледима у апликацији, али можда најједноставији се може наћи на страници за контакте. Као што је раније напоменуто у *ContactsPage.xaml* се приказује листа контаката помоћу компоненте *ListView* која се везује за листу контаката дефинисану у *ContactsViewModel.cs*. Тако се помоћу елемената листе приказују својства контаката (дефинисана помоћу класе 5.1) што се може видети у примеру кôда 5.2.

Пример кôда 5.2: Везивање података у *ContactsPage.xaml*

```
<ViewCell>
    <StackLayout Orientation="Horizontal" BackgroundColor="#17445e">
       <Image Source="{Binding ImageSource}" WidthRequest="40"
              HeightRequest="40" Margin="5, 5, 5, 5"></Image>
       <Label Text="{Binding Name}" Margin="10" TextColor="Azure"></Label>
     </StackLayout>
</ViewCell>
```

Страница за размену порука је представљена помоћу Погледа *MessagingPage.xaml* и њему одговарајућег Поглед-Модела *MessagingViewModel.cs*.

Учитавање контаката је обезбеђено у класи *App.cs* која представља једну од почетних тачака извршавања апликације. Будући да контакти представљају информације специфичне за платформе (пример: за сваку платформу може бити различита локација где се складиште) и да се класа *App.cs* налази у пројекту *MqttChatClient* који представља дељени код, потребно је на неки начин добити наведене информације са свих укључених платформи. У таквим ситуацијама решење долази у виду *DependencyService* класе која омогућава приступ и рад са функционалностима

специфичним за платформе. У примеру *MQChat* апликације, потребне информације представљају контакти и да би се ове информације добиле, било је потребно направити интерфејс, у пројекту *MqttChatClient*, који садржи методе помоћу којих се добијају поменуте информације, затим је потребно додати имплементацију наведеног интерфејса на свакој укљученој платформи (у овом случају само *Android*) и то се може видети у примеру кôда 5.3.

Пример кôда 5.3: Имплементација интерфејса *IContactService* за *Android*

```
[assembly: Dependency(typeof(ContactServiceImplementation))]
namespace MqttChatClient.Droid
{
    public class ContactServiceImplementation : IContactService
    {
       public IEnumerable<PhoneContact> GetAllContacts()
        {...}
       public string GetCurrentPhoneNumberCorrectFormat(string phoneNumber)
        {...}
```

Класу која имплементира наведени интерфејс потребно је регистровати, да би приликом извршавања (енг. *runtime*) *DependencyService* могао да изабере исправну имплементацију, а то се постиже помоћу мета атрибута (у примеру кôда 5.3 то је линија **[assembly: Dependency(typeof(ContactServiceImplementation))]**). У класи *App.cs* функционалност се позива помоћу методе **DependencyService.Get** (пример кôда 5.4).

Пример кôда 5.4: Дохватање података

```
Contacts = DependencyService.Get<IContactService>().GetAllContacts();
```

Битно је напоменути да је *Android* апликацијама потребно одобрити приступ контактима. То се постиже тако што се у датотеци *Android.Manifest* дода дозвола за приступ контактима, а затим је потребно корисника експлицитно питати за дозволу (пример кôда 5.5).

Пример кôда 5.5: Упит за дозволу читања контаката

```
if (ContextCompat.CheckSelfPermission(this, Manifest.Permission.
    ReadContacts) != Permission.Granted)
    {
         ActivityCompat.RequestPermissions(this, new String[] {
```

```
        Manifest.Permission.ReadContacts }, READ_CONTACTS);
    }
```

Кад је реч о размени порука, *MQChat* представља клијентску имплементацију протокола *MQTT*, тј. апликација се претплаћује на тему „сопствениБројТелефона/#“, а објављује поруке на теме облика „пријатељевБројТелефона/сопствениБројТелефона“. Корисник обезбеђује број телефона при пријављивању у апликацију. Будући да бројеви телефона могу бити представљени у разним облицима (пример **(xxx)xxx-xxx или xxx/xxx-xxxx**), бројеви који се користе при објављивању и претплатама на теме су приказани у јединственом интернационалном облику, а то је постигнуто помоћу Гуглове библиотеке *libphonenumber* доступне на урлу https://www.nuget.org/packages/libphonenumber-csharp/. Све претплате и објаве се дефинишу са *тачно једном* квалитетом услуге, тј. *QoS 2*. Детаљи иницијализације се могу видети у примеру кôда 5.6 где клијент покушава успостављање везе са сервером преко сигурног 8883 порта. Као један од аргумената конструктора *MqttClient* прослеђује се *Cert*, који представља сертификат поверљиве странке која гарантује сигурност серверског сертификата. *Cert* је обезбеђен у ресурсима апликације. Један од аргумената је и *MyRemoteCertificateValidationCallback* која представља повратну (енг. *callback*) функцију која се позива приликом *TLS* руковања и пружа додатну могућност валидације серверског сертификата. Приликом успостављања везе, клијент шаље идентификатор који представља његов број телефона. Аргумент *false* представља индикатор да клијент не жели чисту сесију.

Пример кôда 5.6: Иницијализација клијента

```
private void InitializeMQTTClient()
 {
    mqttClient = new MqttClient(BROKER_HOST_NAME, MqttSettings.
                 MQTT_BROKER_DEFAULT_SSL_PORT, true, Cert, null,
                 MqttSslProtocols.TLSv1_2,
                 MyRemoteCertificateValidationCallback);

    mqttClient.Connect(PhoneNumber, null, null, false, 2000);

    if (mqttClient.IsConnected)
    {
         mqttClient.MqttMsgPublishReceived += ClientMqttMsgPublishReceived;

         if (ShouldSubscribe)
         {
```

```
            string topic = string.Format("{0}/#", PhoneNumber);
            mqttClient.Subscribe(new string[] { topic}, new byte[] {
                  MqttMsgBase.QOS_LEVEL_EXACTLY_ONCE });
        }
    }
}
```

Приликом пристизања нове поруке (пример кôда 5.7), исту је потребно сачувати у бази, али и обавестити на неки начин кориснички интерфејс о новопристиглим порукама. *MessagingCenter* омогућава комуникацију међусобно (не)повезаних компоненти. Будући да је догађај за пристизање порука обрађен у класи *App.cs*, она може другим компонетама послати обавештење о том догађају. У примеру кôда 5.7 се види да *MessagingCenter* објављује поруку *newMessageReceived*. Друге компоненте се могу претплатити да ослушкују поруке на ту тему и ускладе понашање у складу са добијеним информацијама.

Пример кôда 5.7: Пријем порука

```
public void ClientMqttMsgPublishReceived(object sender,
            MqttMsgPublishEventArgs e)
    {
        string receivedMessage = Encoding.Default.GetString(e.Message);
        string[] topicLevels = e.Topic.Split('/');

        if (topicLevels.Length == 2)
        {
            string messageSender = topicLevels[1];
            Message message = new Message(){...};
            AppDataBase.SaveItemAsync(message);
            MessagingCenter.Send(this, "newMessageReceived", message);
        }
    }
```

Објављивање нове поруке се остварује путем *Publish* методе (пример кôда 5.8).

Пример кôда 5.8: Објављивање поруке

```
public void PublishNewMessageAsync(Message message)
    {
        string topic = string.Format("{0}/{1}", message.Receiver,
                        PhoneNumber);
        byte[] data = Encoding.UTF8.GetBytes(message.Text);
        if (mqttClient.IsConnected)
```

```
            {
                mqttClient.Publish(topic, data,
                           MqttMsgBase.QOS_LEVEL_EXACTLY_ONCE, false);
                AppDataBase.SaveItemAsync(message);
            }
    }
```

5.2.0.1 Даља унапређења

Да би апликација *MQChat* могла да се користи у комерцијалне сврхе, потребно је обезбедити неколико ствари. Прво, би било неопходно обезбедити одговарајуће сертификате за **SSL/TLS** конекцију, а не самопотписане.
Као што је напоменуто, *Mosquitto* брокер може користити *ACL* помоћу које се остварују рестрикције на претплате и објаве корисника. Пожељно би било дефинисати још један сервер који би служио за регистрацију корисника. Подаци о кориснику би били : корисничко име и лозинка и број телефона. На основу тих података, могла би да се попуни *ACL* и на тај начин би се спречила злоупотреба са темама. Додатни сервер би помогао при провери идентитета мобилног телефона (на пример слањем одговарајућег кода на број). *MQChat* апликација би требала да се допуни са одговарајућим регистрационим кораком.

Глава 6

Закључак

У овом раду приказани су неки аспекти протокола *MQTT* који показују зашто је он један он најпопуларнијих *M2M* протокола данашњице и због тога је највише пажње посвећено начину функционисања протокола. Апликација (*MQChat*) имплементирана за потребе рада је писана у оквиру *.NET* и у таквој ситуацији чини се да употреба *Xamarin*-а представља добар избор.

Приликом рада било је потребно осмислити структуру апликације, тако да се на најбољи начин искористе доступне функционалности. *Xamarin.Forms* се у том погледу показао као изузетно користан и интуитиван скуп алата, који пружа велики асортиман могућности. Приказане су неке од основних функционалности као што су : *Data Binding*, *Messaging Center*, *Dependency Service* итд. које су се показале као изузетно ефикасне и агилне у развоју апликације.

Будући да је протокол *MQTT*, између осталог, дизајниран за окружења са ограниченим изворима енергије, чини да је је његова примена у мобилним апликацијама добар избор с обзиром на то да мобилни телефони раде на батерије. Не треба заборавити ни сва остала својства која га тренутно чине једним од најпопуларнијих протокола у *M2M* комуникацији. Међутим, не треба се искључиво повести причом и популарности протокола *MQTT*. Пре имплентирања решења у било ком систему, потребно размислити о проблемима и потребама датог система и да ли се протокол *MQTT* уклапа у ту причу.

Библиографија

[1] Мирослав Марић, *Оперативни системи*, Универзитет у Београду - Математички факултет, Студентски трг 16, Београд, 2017.

[2] OASIS Standard, *MQTT version 3.1.1*, URL : http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html, 2014.

[3] Lampkin, Valerie and Leong, Weng Tat and Olivera, Leonardo and Rawat, Sweta and Subrahmanyam, Nagesh and Xiang, Rong and Kallas, Gerald and Krishna, Neeraj and Fassmann, Stefan and Keen, Martin and others, *Building smarter planet solutions with mqtt and ibm websphere mq telemetry*, IBM Redbooks, 2012.

[4] Microsoft.Docs, *Xamarin.Forms - Xamarin* , URL : https://docs.microsoft.com/en-us/xamarin/xamarin-forms/, 2018.

[5] Microsoft.Docs, *NuGet*, URL : https://docs.microsoft.com/en-us/nuget/what-is-nuget