Usporedba MQTT i HTTP protokola u IoT okruženju

Kušević, Tomislav

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I

INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

Smjer Računarstvo

USPOREDBA MQTT I HTTP PROTOKOLA U IOT OKRUŽENJU

Diplomski rad

Tomislav Kušević

Osijek, 2022.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 19.09.2022.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Tomislav Kušević
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1136R, 13.10.2020.
OIB studenta:	37391101709
Mentor:	Izv. prof. dr. sc. Damir Blažević
Sumentor:	Izv.prof.dr.sc. Tomislav Keser
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Ivica Lukić
Član Povjerenstva 1:	Izv. prof. dr. sc. Damir Blažević
Član Povjerenstva 2:	Prof. dr. sc. Krešimir Nenadić
Naslov diplomskog rada:	Usporedba MQTT i HTTP protokola u IoT okruženju
Znanstvena grana diplomskog rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak diplomskog rada:	Analizirati MQTTi HTTP protokole te mogućnosti njihove primjene s posebnim nagalskom na IoT okruženje. Izgraditi pokaznu aplikaciju na kojoj će se pokušati praktični primjeri i usporedbe. (Tema rezervirana za: Tomislav Kušević)
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/bodaPostignuti rezultati u odnosu na složenost zadatka: 2 bod/bodaJasnoća pismenog izražavanja: 3 bod/bodaRazina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	19.09.2022.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.* Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 28.09.2022.

Ime i prezime studenta:	Tomislav Kušević
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1136R, 13.10.2020.
Turnitin podudaranje [%]:	13

Ovom izjavom izjavljujem da je rad pod nazivom**: Usporedba MQTT i HTTP protokola u IoT okruženju**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Damir Blažević

i sumentora Izv.prof.dr.sc. Tomislav Keser

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Tomislav Kušević

SADRŽAJ

1.UVOD

Unazad par godina uviđamo sve veći broj uređaja koji zahtijevaju spajanje na internet i pristup internet podacima. Također, pojavljuje se i pojam *Internet of Things* (u daljnjem tekstu: *IoT*) kojem je cilj povezati više uređaja u jednu cjelinu. Sve je veći broj i internet protokola (TCP/IP, HTTP, MQTT, itd.) koji se koristi za samo povezivanje tih uređaja u jednu cjelinu i manipuliranje podacima između istih.

Glavni cilj diplomskog rada je usporediti funkcionalnosti HTTP i MQTT protokola. Možemo reći da oba protokola koriste TCP/IP način povezivanja, zbog tih sličnosti moguće ih je usporediti i dobiti jasniji pregled namjena svakog od protokola.

Kako bi uspješno usporedili ta dva protokola, ovaj rad ćemo bazirati na praktičnom radu koji će nam dati detaljniji pregled svakog od protokola i način funkcioniranja navedenih. Nadalje, dokument ćemo podijeliti u dvije cjeline - prva će biti praktični dio, prethodno opisan, dok je drugi dio teorijski dio u kojem će biti opisan svaki od protokola, njegova namjena i primjena.

1.1. Zadatak diplomskog rada

Zadatak ovog diplomskog rada je izrada aplikacije koja će koristiti HTTP i MQTT protokol kao način povezivanja i komunikacije s udaljenim *serverom* ili drugim uređajima. Na navedenom primjeru dobiti ćemo jasniji uvid u pojedini protokol, njegove prednosti i nedostatke na temelju kojih ćemo razaznati namjenu i svrsishodnost navedenih protokola.

Sama aplikacija je vrlo jednostavna, što nam daje prostora i vremena da se bavimo samim protokolima i njihovom primjernom. Navedena aplikacija ima mogućnost dodavati, uređivati, brisati i čitati bilješke s udaljenog *servera*. Osim toga, svi uređaji će biti obaviješteni kada se dogodi promjena u bazi, tako ćemo simulirati rad *IoT* sustava koji najviše i koristi MQTT protokol i probati pronaći primjenu kod razvoja mobilnih aplikacija.

2. KORIŠTENE TEHNOLOGIJE

U ovom dijelu diplomskog rada opisat ćemo tehnologije koje smo koristili tijekom izrade aplikacije koja će nam omogućiti uvid u namjenu i primjenu HTTP i MQTT protokola. Možemo reći da su oba, HTTP i MQTT, protokola zapravo na aplikacijskom sloju TCP/IP protokola. Kako bi razumjeli što je to aplikacijski sloj i koje još slojeve imamo, u idućem poglavlju ćemo objasniti što je i kako funkcionira TCP/IP protokol.

2.1. Općenito o TCP/IP protokolu

TCP protokol

TCP (*engl. Transmission Control Protocol*) kao protokol na aplikacijskom sloju je zadužen za stvaranje komunikacijskih kanala koje čine mrežu. Također određuje kako će se poruka podijeliti na pakete prije nego što su upućena na odredište koristeći internet. Kada paketi uspješno dospiju na odredište, tada je idući korak sastavljanje paketa u točnom redoslijedu kako bi oni bili razumljivi i validirani.

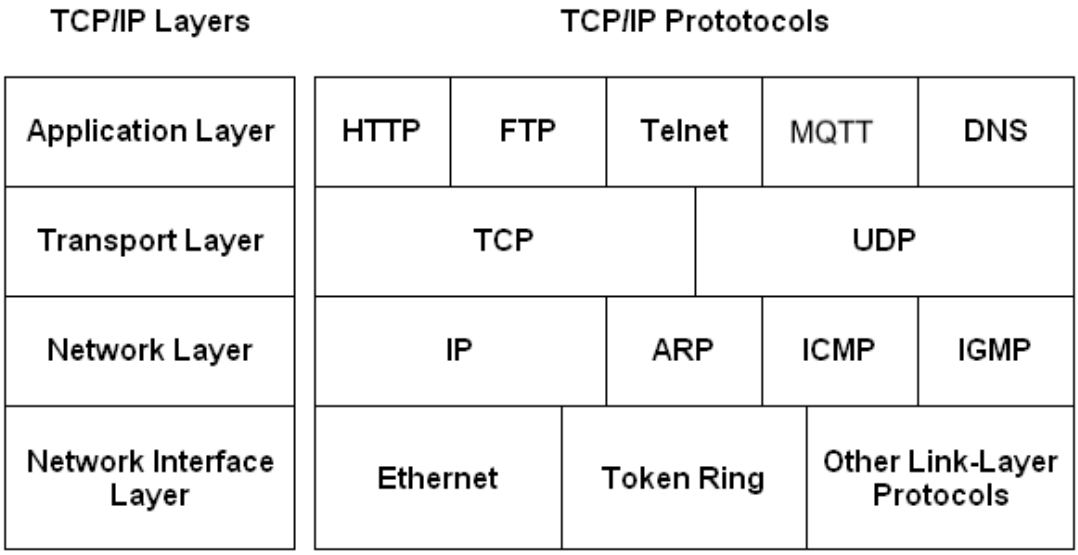
IP protokol

IP (*Internet Protocol*) je mrežni protokol kojem je svrha usmjeravati i adresirati svaki paket kako bi svaki od njih pristigao na svoje odredište. Bazira se na IP adresama zapisanim u zaglavljima paketa (*engl. packet headers*). Najbitniji podaci koji su zapisani u IP strukturi su IP adresa pošiljatelja i IP adresa odredišta.

TCP/IP protokol

Koristeći TCP i IP nastao je takozvani TCP/IP (*Transmission Control Protocol/Internet Protocol*). Prva verzija koja se masovnije koristila je bila IPv4, koji je još i dan danas dominantan protokol koji se koristi kao komunikacija preko interneta, kasnije (2006. godine) nastaje i IPv6 - naprednija verzija protokola koja sve više uzima maha. Za TCP/IP možemo slobodno reći da je već dugo godina standard koji računalima i uređajima pruža i omogućuje udaljenu komunikaciju preko interneta putem paketa. Svaka se poruka šalje i manjim paketima koje najčešće šalji različitim rutama na isto odredište, gdje se sastavljaju i omogućuju da se poruka pročita na odredištu. Kako bi razumjeli kako je to sve moguće, najbolje da opišemo svaki

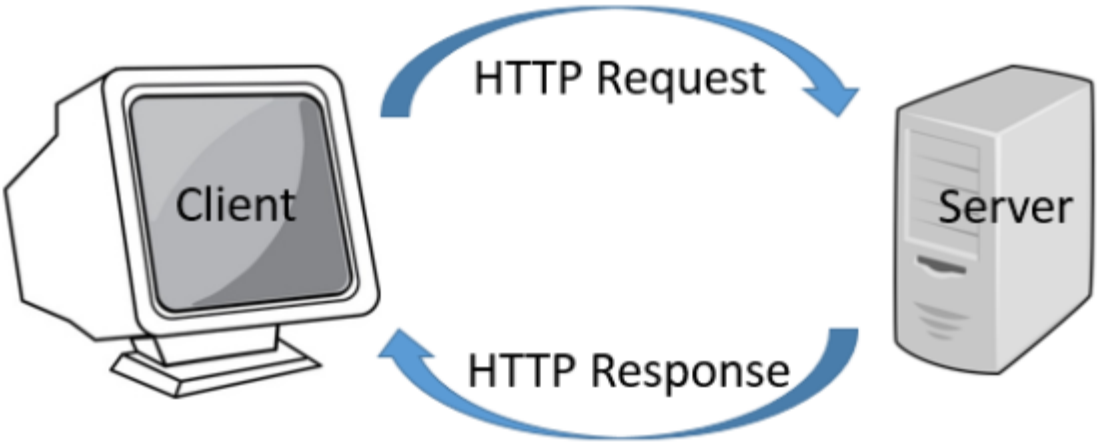
od njih pojedinačno. TCP/IP se sastoji od 4 sloja: aplikacijskog, transportnog, mrežnog i sloja pristupa mreži koji su detaljnije prikazani na idućoj slici.



Slika 2.1. Prikaz slojeva TCP/IP protokola.

2.2. HTTP protokol

HTTP (*Hypertext Transfer Protocol*) protokol je aplikacijski protokol, koji je osnova je podatkovne komunikacije *WEB-a* (*World Wide Web*) i jedan od najkorištenijih protokola na svijetu. Trenutno su u raširenoj uporabi verzije protokola HTTP/1.1 i HTTP/2.0 (postoji još i HTTP/0.9). Komunikacija HTTP protokolom funkcionira po principu zahtjev-odgovor u okviru klijent-poslužitelj arhitekture (slika 2.2). Najčešći klijenti su internet preglednici, mobilne aplikacije, klijentske aplikacije, a najčešći poslužitelji su *WEB* aplikacije.



Slika 2.2. HTTP zahtjev-odgovor u kontekstu klijent-poslužitelj.

Komunikacija HTTP protokolom se odvija tako da klijent uputi HTTP **zahtjev** za nekim resursom (npr. HTML stranica ili operacija pretraživanja) i od poslužitelja dobije HTTP **odgovor**.

Zahtjev (*Request*) sadrži informacije o identifikaciji resursa (odnosno URL resursa), verziju protokola (npr., HTTP/1.1), HTTP metodu (GET, POST, PUT, DELETE, HEAD, OPTIONS ili TRACE), zaglavlja zahtjeva koja definiraju parametre komunikacije (npr. *AcceptLanguage: en, Server: Apache 1.1*) i tijelo zahtjeva s podacima koji se šalju na poslužitelj (npr. korisnički podaci u JSON formatu).

Odgovor (*Response*) sadrži informacije o verziji protokola, statusni kod (npr. 200 OK) koji klijentu zahtjeva daje informacije o procesu obrade zahtjeva, zaglavlja odgovora (npr. *Content-Type: text/html*) i tijelo odgovora s podacima koje poslužitelj vraća klijentu (npr. odgovarajući JSON response).

Mogući statusni kodovi:

100 – Informacije dok obrada zahtjeva još traje

200/201 – Uspjeh

300 – Preusmjeravanje

400 – Pogreška klijenta

500 – Pogreška na poslužitelju

HTTP kao protokol transportnog sloja najčešće koristi TCP protokol, a kao mrežni protokol IP protokol.

2.3. MQTT Protokol

Općenito o MQTT protokolu

MQTT (*Message Queuing Telemetry Transport*) jedan je od najstarijih M2M (*Machine to Machine*) komunikacijskih protokola. Andy Stanford-Clark-a iz "IBM-a" i Arlen Nipper-a iz tvrtke "Arcom Control Systems Ltd" su 1999. godine razvili MQTT. Glavna karakteristika MQTT protokola su lagane poruke izrazito pogodne za *IoT* sustave. MQTT se sastoji od klijenta i posrednika koji međusobno komuniciraju tako da klijent objavljuje poruke, dok poslužitelj iste poruke prosljeđuje do klijenata koji su se pretplatili na tu temu. Svaka se poruka objavljuje na adresi koja je poznata kao tema. Klijenti se mogu pretplatiti na više tema i primati svaku poruku objavljenu za svaku temu (*topic*).

Najbitnije funkcije MQTT protokola

Najčešće kada govorimo o MQTT protokolu spominjemo neke od osnovnih funkcija koje su nužne kako bi se klijent (u ovom slučaju uređaj) uspješno primio podatke od izvora (u ovom slučaju brokera), a to su:

- *connect* (povezivanje)
- *subscribe* (predbilježavanje)
- *publish* (objavljivanje)
- *unsubscribe* (odbilježavanje)

Funkcija Connect (povezivanje)

Funkcija *connect* koristi se kako bi povezala klijenta na određeni izvor podataka (u daljnjem tekstu broker) koji osim povezivanja govori izvoru i informacije koje su od velike važnosti i brokeru kako bi on znao tko/kako/zašto je povezan na njegov izvor informacija. Neke od informacija koje klijent (uređaj) mora poslati brokeru tijekom akcije povezivanje su:

- *ClientID* - to je jedinstvena identifikacijska oznaka koja predstavlja svaki uređaj kao

jedinstveni kako bi se moglo razlikovati spojene uređaje na broker. Ako on nema vrijednost tada kažemo da je uređaj povezan anonimno i informacije o tom uređaju neće biti spremljene u sesiju

- *CleanSession* - predstavlja boolean vrijednost, koja govori brokeru informaciju da u slučaju da ima nekih zaostalih ili nepročitanih poruka - treba li mu ih poslati nakon što se uređaj poveže

- *KeepAlive* - predstavlja informaciju o tome koliko dugo smije održavati konekciju živom (između klijenta i brokera) ako nema izmjene poruka među njima.

- *Username and Password* (*optional*) - Klijent postavlja svoje podatke (e-mail i zaporku) kako bi se postigla veća razina sigurnosti

- *WillMessage* (*optional*) - Klijent može postaviti određenu poruku koju želi primiti od brokera ukoliko dođe do neočekivanog prekida veze

Funkcija *Subscribe* (predbilježavanje)

Funkcija *subscribe* se koristi kako bi se klijent predbilježio na određenu temu (u daljnjem nastavku *topic*). Neke od informacije koje se moraju poslati brokeru kod pozivanje funkcije *subscribe* su:

- *Topic Name* **-** tema na koju se klijent želi predbilježiti

- *QoS Level* **-** predstavlja kvalitetu servisa koja ima utjecaj na primanje poruka

Razlikujemo iduće 3 vrste QoS Level-a:

1.) 0 — A*t most once* - predstavlja garanciju da će se poruka maksimalno jednom isporučiti. Naziva se još i "Ispucaj i zaboravi" zbog toga što možemo računati da je komunikacija stabilna i ne očekuju se prekidi veze

2.) 1 - *At least once* - predstavlja garanciju da će se poruka barem jednom isporučiti. Koristi se kada se duplicirane poruke mogu dopustiti - najčešće u uporabi

3.) 2 - *Exactly once* - predstavlja garanciju da će se poruka točno jednom isporučiti. Koristi se ako aplikacija (klijent) ne smije primiti istu poruku više od jednom (duplikat)

No, podižući razinu kvalitete (QoS Level) komunikacije dolazi do sporijih performansi i obrnuto.

Funkcija *Publish* (slanje)

Funkcija *Publish* se koristi kako bi se određena poruka poslala na određeni *topic*. Neki od informacija koje je potrebno poslati kod pozivanja metoda su:

- *Topic Name* - tema na koju želimo poslati poruku
- *QoS Level* - predstavlja kvalitetu servisa koja ima utjecaj na primanje poruka
- *Payload* - podaci koje želimo poslati
- *Retain flag* - predstavlja zastavicu koja je spremljena na brokeru kao posljednju postavljenu poruku koju klijenti koji se povežu mogu dohvatiti

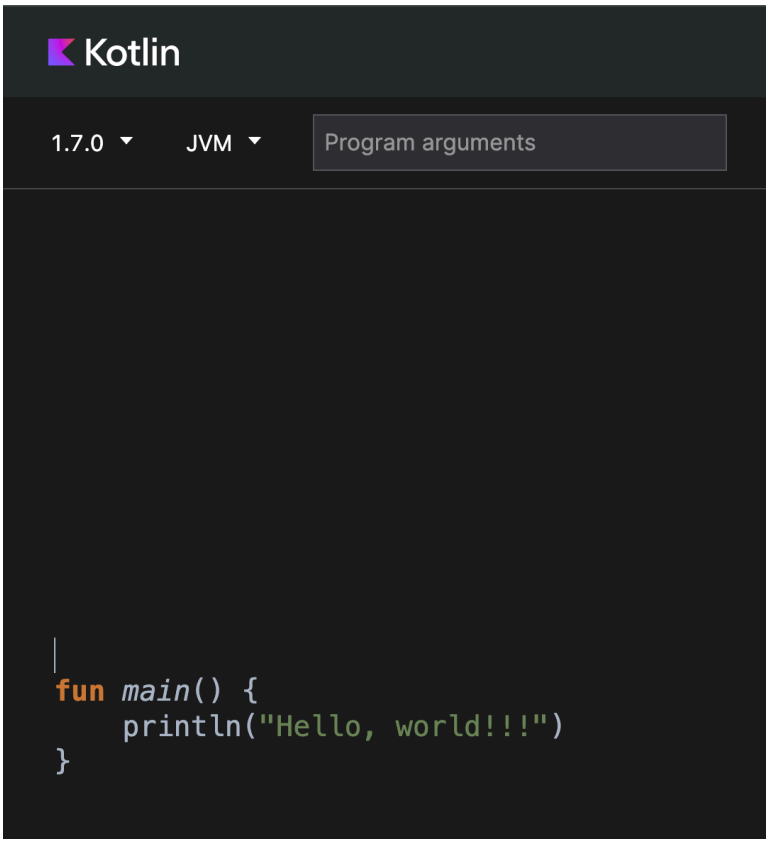
Funkcija *Unsubscribe* (odbilježavanje)

Funkcija *Unsubscribe* se koristi kako bi se klijent odbilježio na određen *topic*. Neke od informacije koje se moraju poslati brokeru kod odbilježavanja funkcije *Unsubscribe* su:

- *Topic Name* - tema s koje se klijent želi odbilježiti
- *QoS Level* - predstavlja kvalitetu servisa koja ima utjecaj na primanje poruka

2.4. Kotlin

Kotlin je višeplatformski programski jezik koji je sažet, siguran, interoperabilan i vrlo ga je lako koristiti uz druge alate. To je statički tipizirani programski jezik koji radi na Java virtualnom stroju i također se može prevesti u JavaScript izvorni kod, što ga čini razumljivim velikom broju developera koji razvijaju aplikacije u programskim jezicima: Java, JavaScript ili TypeScript. Razvijen u tvrci "JetBrains", poznate po razvoju moćnih integriranih razvojnih sučelja za programski jezik Java pod nazivom "IntelliJ IDEA". Danas se Kotlin vodi kao preferirani programski jezik za razvoj Android aplikacija - što je potvrdio i Google.

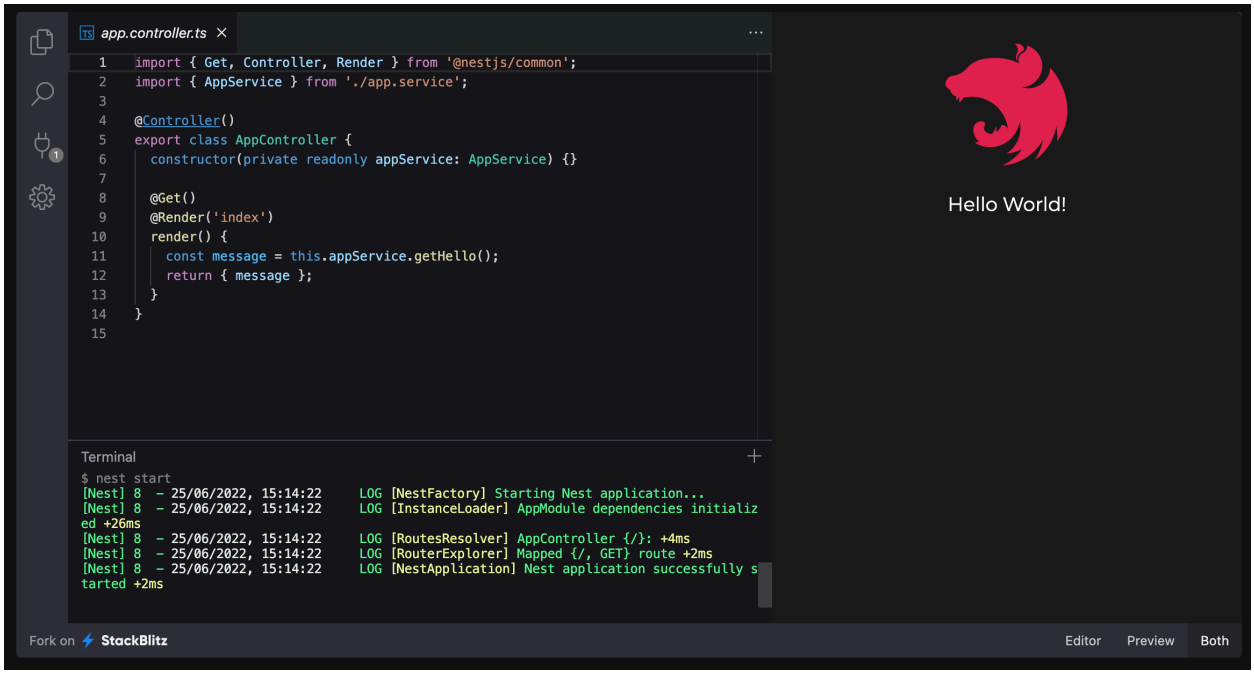


Slika 2.3. Primjer *Hello World* koda u Kotlinu.

2.5. NestJS

NestJS je platforma (u nastavku: *framework*) baziran na Node.js platformi koji je previđen za upotrebu s TypeScript (JavaScript) jezikom kojem je cilj što brži i jednostavniji razvoj skalabilnih aplikacija na poslužiteljskoj (*server*) strani.

Aplikacije koje su se razvijaju na NestJS platformi bazirane su na komunikacijskim paketima kao što su Express ili Fastify. Nest je nova Node.js platforma koja ne samo da oponaša nego i ispravlja nedostatke prethodnih Node.js verzija. Kada započnete novi Node.js projekt, NestJS je puno bolji izbor od ExpressJS-a, jer je zamišljena arhitektura projekta već definirana unaprijed s nekoliko jednostavnih komponenti (kontroleri, moduli i servisi). To čini podjelu aplikacija u mikroservise jednostavnijim.



Slika 2.4. Primjer *"Hello World"* rute u NestJS kontroleru.

2.6. PostgreSQL

PostgreSQL je moćan sustav objektno-relacijskih baza podataka otvorenog koda koji koristi i proširuje SQL jezik u kombinaciji s mnogim značajkama koje sigurno dohvaćaju, obrađuju i pohranjuju moguću znatnu količinu podataka. Počeci PostgreSQL-a datiraju iz 1986. godine kao dio projekta POSTGRES na Sveučilištu Kalifornija, u gradu Berkeley-u.

PostgreSQL je stekao snažnu reputaciju zbog svoje dokazane arhitekture, pouzdanosti, integriteta podataka, robusnog skupa značajki, proširivosti i predanosti zajednice otvorenog koda koja stoji iza softvera da dosljedno isporučuje učinkovita i inovativna rješenja. PostgreSQL radi na svim glavnim operativnim sustavima. Nije iznenađenje da je PostgreSQL zbog svojih svojstava postao izbor mnogih. PostgreSQL smo koristili kao bazu podataka u koju spremamo bilješke i korisnike. U nastavku se nalazi popis tablica unutar baze podataka korištene u projektu (Sl. 2.5.).

```
Tomislavs-MBP:notesAPI tkusevic$ psql notes
sed: illegal option -- r
usage: sed script [-Ealn] [-i extension] [file ...]
       sed [-Ealn] [-i extension] [-e script] ... [-f script_file] ... [file ...]
psql (14.5)
Type "help" for help.

notes=# \dt
              List of relations
 Schema |       Name       | Type  |  Owner
--------+------------------+-------+----------
 public | migrations       | table | postgres
 public | notes            | table | postgres
 public | typeorm_metadata | table | postgres
 public | users            | table | postgres
(4 rows)
```

Slika 2.5. Popis tablica baze podataka.

Kako bi uspjeli dohvatiti korisnike i bilješke, navodi se primjer dohvaćanja takvih podataka iz baze koja je namještena za lokalnu uporabu (Sl. 2.6.).

```
notes=# select * from user;
   user
----------
 tkusevic
(1 row)

notes=# select * from notes;
                  id                  | title | description |         createdAt          |         updatedAt          |               userId
--------------------------------------+-------+-------------+----------------------------+----------------------------+--------------------------------------
 869bcc71-196a-4b5c-ae3c-0396566ecf00 | Title | Hello World | 2022-08-21 20:36:33.246692 | 2022-08-21 20:36:33.246692 | f7abe9c8-ae17-4d5d-b54f-398b5e3e2f2a
(1 row)
```

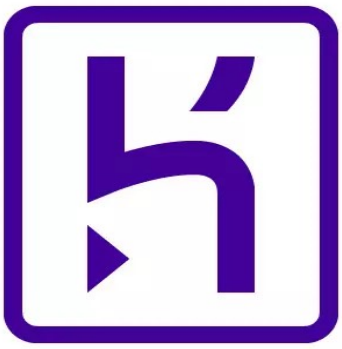
Slika 2.6. Primjer dohvaćanja korisnika i bilješki iz baze.

Da bi imali pristup podacima, bazu je potrebno podići na server koji nije lokalno na računalu, nego joj moramo omogućiti pristup preko interneta. Zato smo koristili Heroku kao oblik dostavljanja servera i baze podataka na određenoj internet lokaciji. Heroku ima opciju da prepozna korištenje baze podataka, samo je potrebno odabrati Heroku Postgres kao oblik dodatka na serveru.

2.7. Heroku

Heroku je platforma kao usluga (*PaaS - Platform as a Service*) koja isporučuje alate koji omogućuju razvoj softvera. Heroku, kao usluga, omogućuje korisnicima brzu implementaciju, izradu, upravljanje i skaliranje aplikacija bez znanja o infrastrukturi, već uz par klikova podiže svoj server dostupan na internetu. Djeluje kao posrednik između korisnika i mjesta gdje je aplikacija podignuta (*host*).

Sve aplikacije koje se podižu na Heroku, uz pomoć AWS-a (*Amazon Web Service*) su podignute na *host*. AWS smatramo pružatelj infrastrukture kao usluge (*IaaS - Infrastructure as a Service*). AWS upravlja računalnim resursima i značajno skraćuje vrijeme podizanja sustava na *host* tijekom razvoja aplikacije.
Balansiranje opterećenja, odabir operativnog sustava, poslužitelji, mreže, pohrana, bilježenje i praćenje poslužitelja, praćenje ispravnosti aplikacije, uspostavljanje organizacije spremnika - svime upravlja AWS (korisnik ne mora imati znanje o navedenom kako bi uspješno podigao server na određeni *host*).



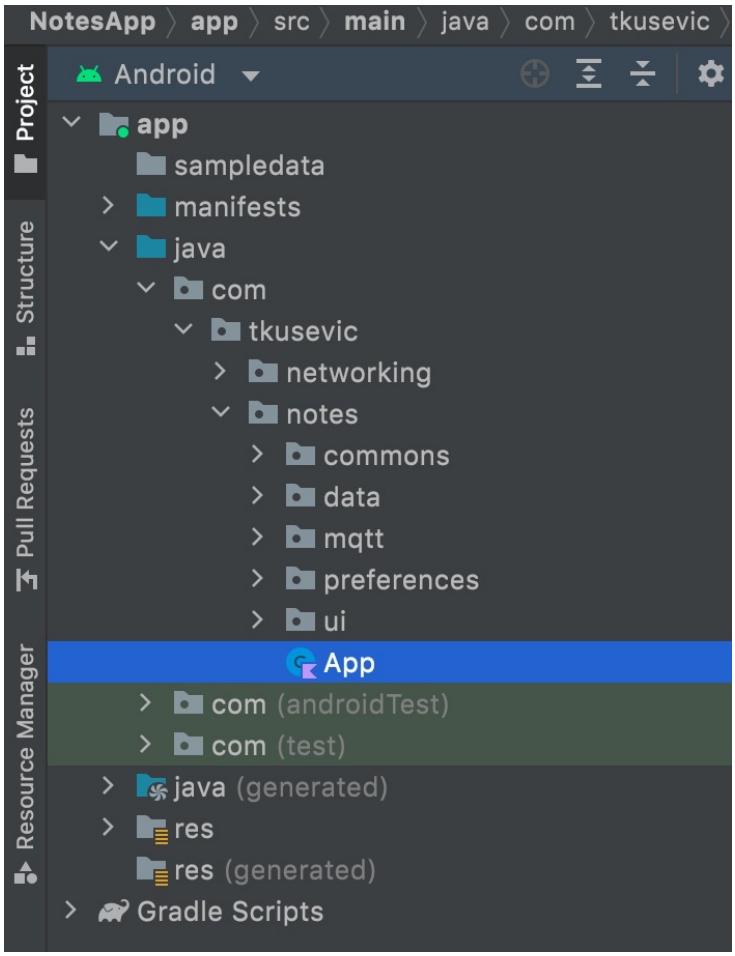
Slika 2.7. Logo platforme Heroku.

2.8. Android Studio

Android Studio je službeno razvojno okruženje (*IDE*) za razvoj Android mobilnih aplikacija, temelji se na IntelliJ IDEA. Uz IntelliJ kao uređivač koda i alate za razvojne programere, Android Studio nudi još i mnogo više alata koje povećavaju produktivnost pri izradi Android aplikacija, kao što su:

- Sustav temeljen na Gradle-u
- Uz povezivanje na uređaje, pruža i mogućnost pokretanja aplikacija na emulatoru
- Okruženje koje pruža razvoj aplikacija za bilo koji Android uređaj (mobitel, TV i sl.)
- Gotovi predlošci koji ubrzavaju razvoj i popravke nad samom aplikacijom
- Sustav za brže i efikasnije testiranje aplikacije
- Alat koji pomaže pri pisanju koda, ispravcima grešaka, poravnanju koda i sl.
- Kotlin, Java i C++ podršku
- Pomoć pri objavi aplikacija na Google Play sustav

Struktura Android Studio-a je stablastog oblika (Sl. 2.8.).



Slika 2.8. Struktura projekta u Android Studio-u.

Unutar svake mobilne Android aplikacije pisane u Android Studio-u nalaze se sljedeće grupe:

• *manifest* – grupa u kojoj je sadržana AndroidManifest.xml datoteka

• *java* – grupa u kojoj je sadržan programski kod najčešće pisan u programskim jezicima Java i Kotlin

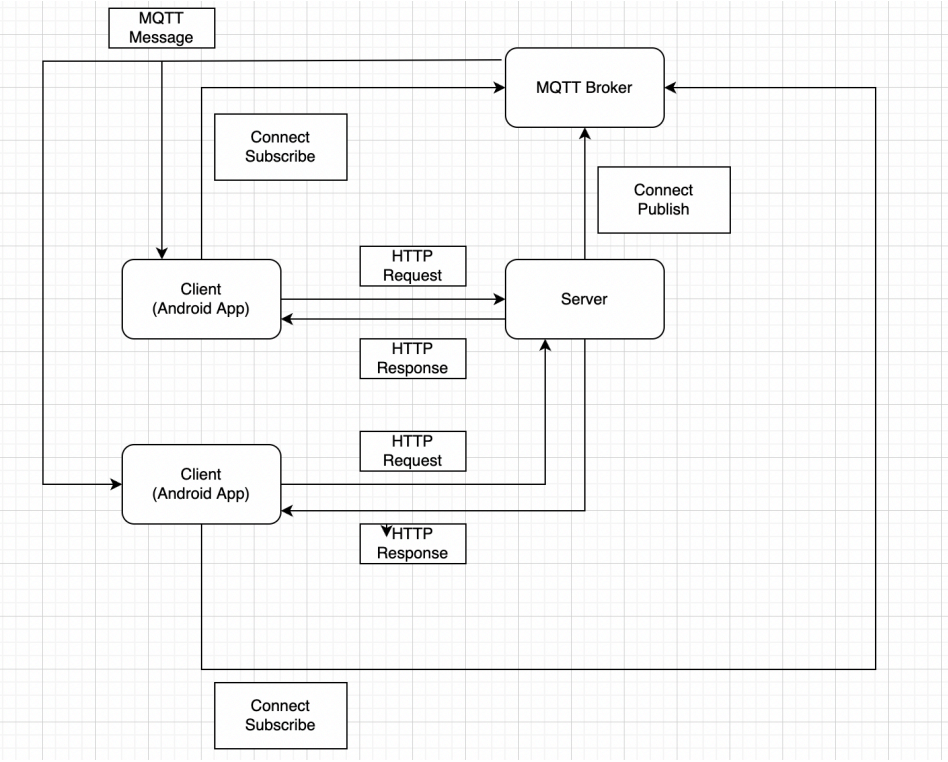
• *res* – grupa u kojoj se nalaze resursi vezani za aplikaciju, kao što su slike, boje, izgledi pojedinih prozora i sl.

3. IMPLEMENTACIJA PROGRAMSKOG RJEŠENJA

Implementaciju programskog rješenja možemo podijeliti na 2 osnovna dijela:

- Razvoj dijagrama toka
- Razvoj servera i baze podataka
- Razvoj mobilne (Android) aplikacije koja komunicira preko HTTP protokola (*Request & Response*) i MQTT protokola (*Publish & Subscribe*) sa *serverom*

3.1. Razvoj dijagrama toka



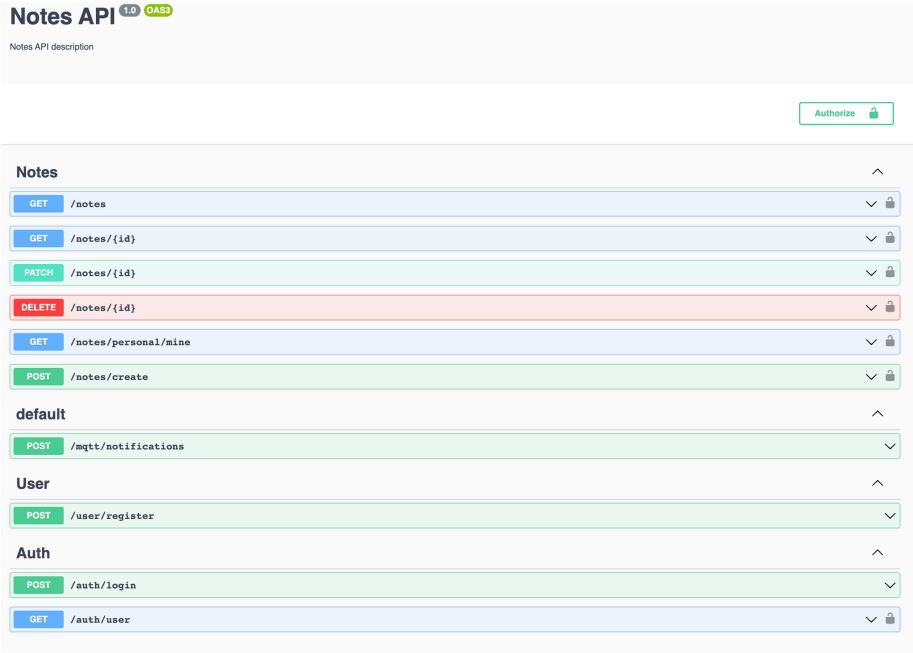
Slika 3.1. Prikaz dijagrama toka projekta.

Na dijagramu (Sl. 3.1.) klijenti (Android aplikacije) se povezuju sa serverom preko HTTP protokola zahtjevima i čekaju odgovore kao rezultat svakog zahtjeva, a prema MQTT Brokeru nakon povezivanja imaju mogućnost predbilježavanja na određenu temu. Također, server ima mogućnost spajanja na MQTT Broker i funkcijom *Publish* ima mogućnost slanja svim klijentima određenu poruku.

3.2. Razvoj poslužitelja i baze podataka

Kako bi naša aplikacija funkcionirala bilo je potrebno prvotno kreirati i razviti preduvjete da spremimo određene podatke u bazu, klijenti (uređaji) imaju pristup tim podacima te ih mogu mijenjati, pregledavati, dodavati i brisati. Kao bazu podataka smo koristili ranije navedenu PostgreSQL bazu podataka (lokalno, kasnije podignutu bazu podataka na serveru putem Heroku servisa). Sam pristup aplikaciji nalazi se na internet lokaciji: https://notes-mqtt.herokuapp.com/, a dokumentacija (za koju smo koristili proširenje po imenu *Swagger*) nalazi se na internet lokaciji: https://notes-mqtt.herokuapp.com/api.

Pristup podacima je omogućen na različitim rutama i korištenjem različitih HTTP metoda. Sve rute kojima se može pristupiti aplikaciji se nalaze u dokumentaciji i prikazane na idućoj slici (Sl. 3.1.).



Slika 3.2. Prikaz ruta za pristup serveru.

Ukratko ćemo opisati neke od najviše korištenih ruta koje omogućuju klijentima (uređajima) pristup podacima sa servera:

- POST **/user/register** - ruta koja se koristi za registraciju korisnika
- POST **/auth/login** - ruta koja se koristi za prijavu korisnika
- GET **/auth/user** - dohvaćanje podataka trenutno prijavljenog korisnika
- GET **/notes** - dohvaćanje svih bilješki
- GET **/notes/:id** - dohvaćanje točno određene bilješke po identifikacijskom znaku *(id-u)*
- DELETE **/notes/:id** - brisanje točno određene bilješke po identifikacijskom znaku *(id-u)*
- PATCH **/notes/:id** - uređivanje točno određene bilješke po identifikacijskom znaku *(id-u)*

3.3. Razvoj mobilne aplikacije

Android aplikacija se razvijala u programu zvanom Android Studio. Svaki klijent ima opciju otvoriti mobilnu aplikaciju na svom Android uređaju, registrirati se i time mu je omogućen pristup aplikaciji. Svaki korisnik je nazvan još i klijent. U ovom poglavlju opisat ćemo funkcionalnosti aplikacije, mogućnosti koje korisnik ima i implementaciju MQTT i HTTP protokola. Za početak možemo opisati i prikazati funkcionalnosti mobilne aplikacije.

3.4. Funkcionalnosti mobilne aplikacije

Razvijena Android aplikacija sastoji se od:

- prozora pokretanja (*Splash screen*)
- prozora registracije (*Register screen*)
- prozora prijave (*Login screen*)
- prozora bilježaka (*Notes screen*)
- prozora prikaza detalja (*Notes details screen*)

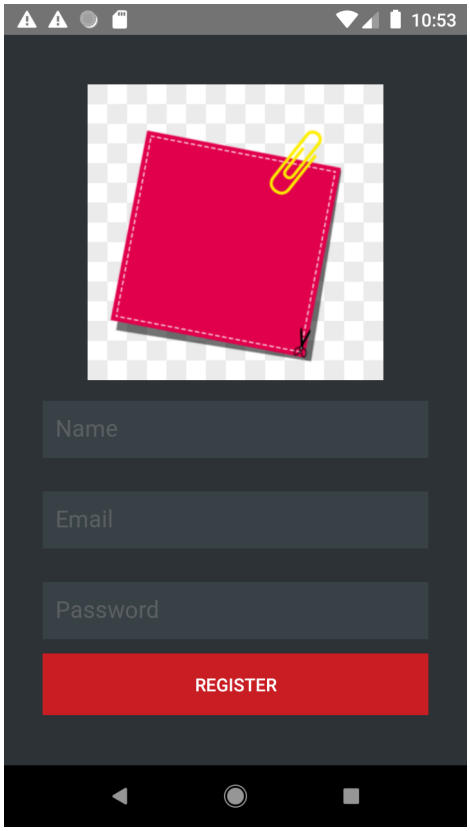
Prozor pokretanja

Funkcionalnosti mobilne aplikacije kreće od prvog početnog prozora - prozora pokretanja (*Splash screen*) na kojem se vrši provjeravanje memorije u uređaju kako bi se utvrdilo postoji li ulogirani user na mobilnom uređaju. Ako se korisnik već registrirao i prijavio nekad, njegov identifikacijski broj će biti zapisan u memoriju, te će ga aplikacija preusmjeriti direktno u aplikaciju, ako korisnik nije prethodno ulogiran (ili je obrisao internu memoriju aplikacije) bit će preusmjeren na prozor registracije.

Prozor registracije

Ako korisnik nema registrirani račun, taj proces mu je omogućen na prozoru registracije (Sl. 3.2). Potrebni parametri registracije su *e-mail* i zaporka. Osim što *e-mail* mora biti validan *e-mail*, zaporka se mora sastojati od barem 6 znakova kako bi bila važeća. Za registraciju, na serveru se koristi paket JWT (*JSON Web Token*) i JWT strategija koja omogućuje proces autentikacije (registracije i prijave).

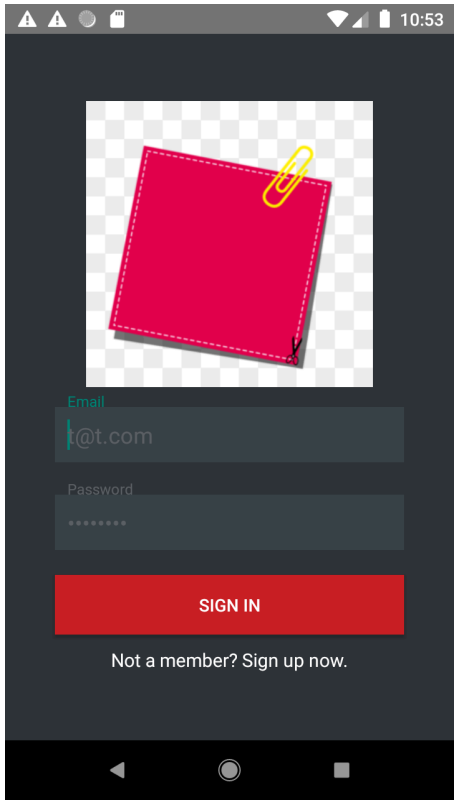
Također, nakon uspješne registracije korisnik će dobiti u odgovoru servera token s kojim ima mogućnost pristupati ostatku poziva na server (ruta). Unutar svakog poziva, klijent mora poslati token kako bi dobio valjajući odgovor *servera*, ali isto tako i *server* zna koji točno klijent želi pristup podacima i ima li pravo na određenu akciju koju pokušava izvršiti. *E-mail* i zaporka su spremljeni u bazu kako bi kasnije znali validirati određenog korisnika tijekom prijave u sustav.



Slika 3.3. Prozor registracije.

Prozor prijave

Ukoliko korisnik ima napravljen račun unutar aplikacije tada može na prozoru prijave napraviti proces prijave. Za proces prijave potrebni su *e-mail* i zaporka, prethodno napravljenog računa na prozoru registracije. Ako se proces prijave uspješno provede, korisniku je omogućen pristup aplikaciji, točnije prozoru bilježaka (listi bilježaka).



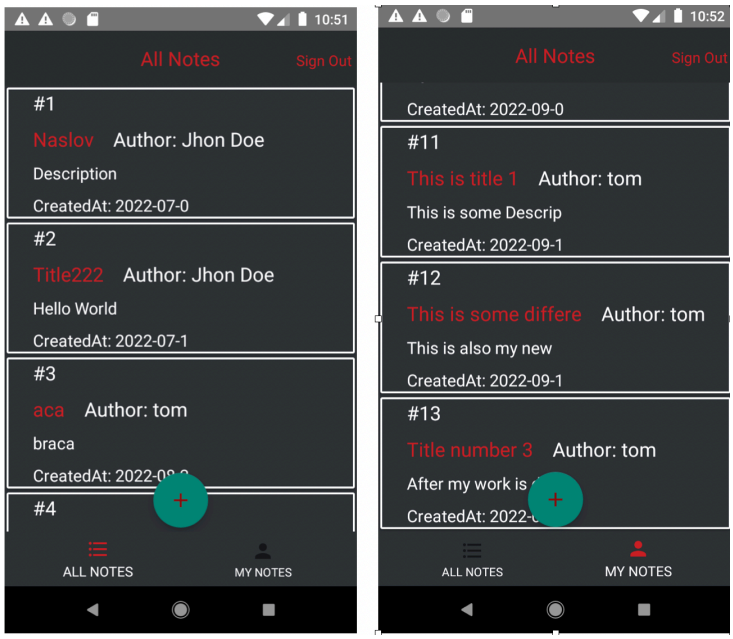
Slika 3.4. Prozor prijave.

Prozor bilježaka

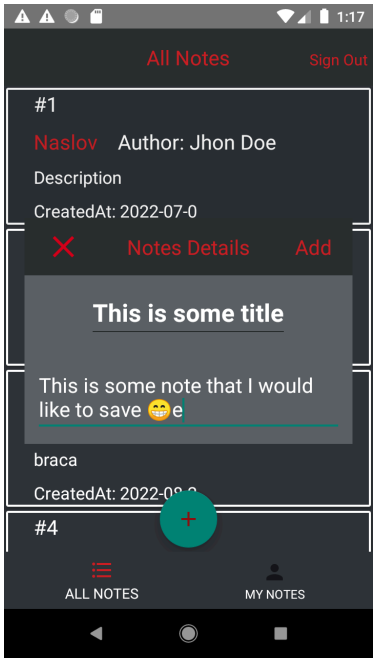
Glavni dio aplikacije je prozor bilježaka. Unutar tog prozora nalaze se dvije kartice: kartica svih bilježaka (*All notes*) i kartica korisnikovih bilježaka (*My notes*). Na prozoru bilježaka nalazi se popis svih bilježaka u kojem imamo mogućnosti vidjeti sve bilješke u kartici sve bilješke (Sl. 3.4. lijevo), dok u kartici korisnikovih bilježaka korisnik ima pregled bilježaka koje je on unio u aplikaciju (Sl. 3.4. desno). Unutar prozora bilježaka, prvo se dohvaćaju bilješke sa servera HTTP pozivom i registrira se MQTT konekcija. Aplikacija se predbilježava na temu (u daljnjem tekstu *topic*) pod nazivom *notes* koja sluša sve poruke poslane na *topic notes* - to omogućuje korisniku na određena promjena unutar baze (dodana ili obrisana bilješka) da se podaci ponovno povuku sa *servera* te da se lista osvježi. Tako imamo mogućnost osvježavanja liste podataka bez potrebe za ručnim osvježavanjem. Tako da tu možemo vidjeti prednosti korištenja MQTT protokola unutar mobilne aplikacije.

Korisniku se pruža mogućnost brisanja vlastitih bilježaka direktno s liste bilježaka. Omogućeno

je i dodavanje bilješke pritiskom na znak "+" donjem dijelu ekrana, nakon koje se pojavljuje prozor u kojem je moguće dodati naslov i tekst bilješke (Sl. 3.6.).



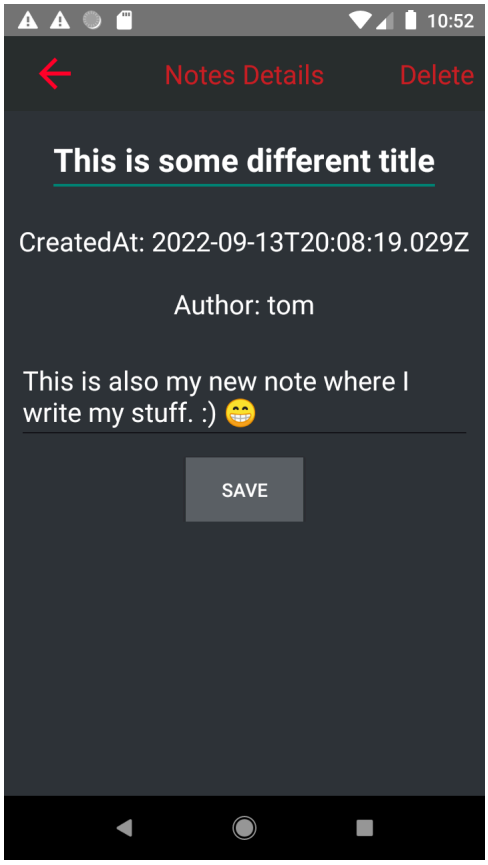
Slika 3.5. Prozor bilježaka (sve i vlastite bilješke).



Slika 3.6. Prozor dodavanja bilježaka.

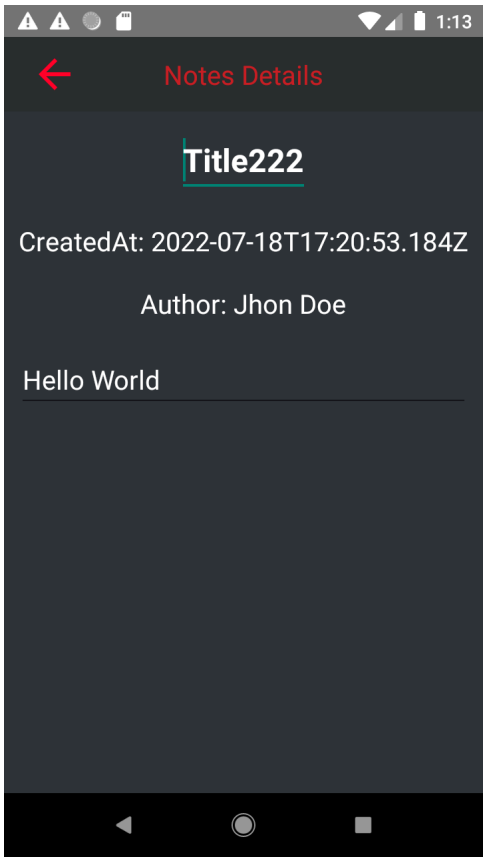
Prozor prikaza pojedine bilješke

Korisnik osim popisa bilježaka, ima mogućnost pregleda svih detalja određene bilješke (Sl. 3.5). Klikom na određenu bilješku, otvara se novi prozor s prikazom svih detalja. Kod prikaza detalja prikazujemo sljedeće podatke: naslov bilješke, tekst bilješke, datum dodavanja bilješke, ime autora bilješke. Korisnik ima mogućnost i obrisati bilješku (ako je on autor iste) pritiskom na tekst *delete* u gornjem desnom kutu. Osim brisanje, također korisnik ima mogućnost izmijeniti naslov i tekst bilješke (Sl. 3.7.).



Slika 3.7. Prozor prikaza pojedine bilješke (korisnik je autor).

U slučaju da trenutni korisnik nije autor bilješke koju je otvorio, tada mu mogućnosti *save* i *delete* nisu prikazane na prozoru (Sl. 3.8.).



Slika 3.8. Prozor prikaza pojedine bilješke (korisnik nije autor).

Tako osiguravamo da korisnik ne može obrisati tuđu bilješku, nego samo svoje vlastite. Osim što to nije omogućeno korisniku u mobilnoj aplikaciji, također je dodana validacija na *serveru*. Time osiguravamo validaciju s dvije strane, što čini naš sustav sigurniji.

4. USPOREDBA MQTT I HTTP PROTOKOLA

Razvojem ove aplikacije možemo reći da MQTT ima svoju primjenu unutar razvoja mobilnih aplikacija. Kao što smo već i naveli, HTTP protokol se već dugo godina i u vrlo velikom postotku koristi za dohvaćanje podataka s interneta (najčešće *servera*). Iako se MQTT protokol najviše koristi u *IoT* sustavima, sada imamo primjer korištenja takvog oblika protokola i u svijetu mobilnih aplikacija. Radi lakše usporedbe, koristit ćemo iduće tri tablice kako bi prikazali određene informacije i podatke o korištenju oba protokola:

Tab. 4.1. Tablica opće usporedbe MQTT i HTTP protokola.

	MQT	HTTP
Ime	MQ Telemetry Transport	Hyper Text Transfer Protocol
Arhitektura	Slanje/predbilježavanje (Publish/Subscribe)	Zahtjev/odgovor (Request/Response)
Odredište zahtjeva	Tema (topic)	URL adresa
Baziran na protokolu	TCP/IP	TCP/IP
Sigurnost veze	TLS + ime/zaporka	TLS + ime/zaporka
Tip konekcije	Status konekcije poznat	Status konekcije nepoznat
Tip slanja poruke	Asinkron, baziran na događajima	Sinkron

	MQT	HTTP
Red čekanja poruke	Broker je u mogućnosti pamtiti zahtjeve	Ovisno o implementaciji unutar aplikacije
Veličina zaglavlja	Minimalno 2 byte, može biti binaran	Minimalno 8 byte, tekstualnog oblika (kompresija moguća)
Veličina poruke	Maksimalno 256MB	/
Tip poruke	Nije ograničenog tipa, najčešće binaran	Tekst (binaran tip je enkodiran s Base64)
Distribucija poruka	Jedan naprema više	Jedan naprema jedan
Pouzdanost	0 - pošalji i zaboravi 1 - pošalji barem jednom 2 - pošalji jednom i samo jednom	Ovisno o implementaciji unutar aplikacije

Tab. 4.2. Tablica usporedbe veličine korištene memorije funckija.

Naziv funkcije	MQTT	HTTP
Uspostava konekcije	5572 B	2261 B
Prekid konekcije	376 B (opcionalno)	0 B
Slanje poruke	388 B	3285 B
Ukupno za 1 poruku	6336 B	5546 B
Ukupno za 10 poruka	9829 B	55,046 B
Ukupno za 100 poruka	44,748 B	554,600 B

Tab. 4.3. Tablica usporedbe veličine korištene memorije u odnosu na broj poruka.

Broj poruka	MQTT - vrijeme odgovora (ms) (QoS 1)	HTTP vrijeme odgovora (ms)
1	113	289
100	47	289
1000	43	289

5. ZAKLJUČAK

Usporedbom HTTP i MQTT protokola na jednostavnoj aplikaciji možemo vidjeti prednosti, nedostatke i predviđeni rad protokola. Možemo reći kako bi HTTP protokol koristili kada bi morali slati veliku količinu podataka (npr. veliki *JSON*) za spremanje određenih podataka u bazu i sl. Svaki put kada otvorimo HTTP poziv, dobijemo podatke, konekcija se zatvori.

S druge strane, MQTT protokol je odličan "lagani" protokol koji nam omogućuje da kad jednom povežemo dva uređaja (pretplatimo se na *topic*) moguća je njihova komunikacija bez zatvaranja iste. Dokle got je neki uređaj pretplaćen na *topic* - poruke će mu kontinuirano pristizati.

Osim dvosmjerne komunikacije (uređaj može i primati poruke na *topic*, ali i slati na isti *topic*) MQTT nam omogućuje i komunikaciju s više uređaja. Recimo da imamo 2 uređaja u kući koji stalno šalju promijenjene podatke (npr. termostat, termohigrometar) koji mjere temperaturu i vlagu - takav uređaj može slati podatke na određeni topic svim ostalim uređajima koji su spojeni na isti. Samim time potrebno je poslati jednom poruku na topic i svi ostali pretplaćeni uređaji će tu poruku primiti (tzv. "*broadcast* podataka").

MQTT se također može koristiti kao i protokol za slanje obavijesti prema uređajima: Uzmimo u obzir da imamo aplikaciju koja se bavi pisanjem i uređivanjem dokumenata/bilježaka. Kada bi dvoje ili više ljudi uređivalo isti dokument moglo bi doći do race-conditiona nad podacima gdje bi jedan korisnik spremio dokument nakon promjena, a drugi korisnik te promjene ne bi uhvatio jer je otvorio dokument prije uređivanja i pregazio promjene prethodnog korisnika - u toj situaciji nam MQTT može biti od pomoći. Svaki korisnik prilikom otvaranja dokumenta bi se pretplatio na topic, u slučaju promjene nad dokumentom - svi korisnici bi dobili obavijest da je dokument promijenjen i da povuku posljednje promjene. Tako smo uklonili race-condition podataka nad dokumentom.

Neku veću implementaciju MQTT protokola vidim dakle u *IoT* industriji (komunikacija uređaja), ali i kod razvoja aplikacija (neki oblik obavijesti).

LITERATURA

[1] B. Phillips, C. Stewart, K. Marsciano, “Android Programming: The Big Nerd Ranch Guide (3rd Edition)”, USA, 2017.

[2] Egli, Peter. (2017). “MQTT - Message Queueing Telemetry Transport Introduction to MQTT, a protocol for M2M and IoT applications”

[3] Bensakhria, Ayoub. (2020).” IoT - Communicating with devices: Introduction to MQTT and HTTP”.

[4] Grigorik, I. : “High Performance Browser Networking ", O'Reilly Media, rujan 2013

[5] Fielding, R.; Reschke, J. : “Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, Internet Engineering Task Force (IETF), lipanj 2014

[6] M. Šušnja, “PRIMJENA MQTT PROTOKOLA U INTERNETU STVARI", Specijalistički diplomski stručni, Sveučilište u Splitu, Sveučilišni odjel za stručne studije, Split, 2020. Dostupno na: https://urn.nsk.hr/urn:nbn:hr:228:047974

[7] P. Šimec Upravljanje IoT uređajima pomoću MQTT poruka, Diplomski rad, Sveučilište u Zagrebu, Fakultet organizacije i informatike

[8] "GeeksForGeeks; Transport Layer responsibilities”. https://www.geeksforgeeks.org/transport-layer-responsibilities/ (pristupljeno 11. kolovoza 2022.).

[9] "MQTT Vs. HTTP for IoT“. https://www.hivemq.com/blog/mqtt-vs-http-protocols-in-iot-iiot/ (pristupljeno 12. kolovoza 2022.)

[10] "Android and MQTT: A Simple Guide“. https://medium.com/swlh/android-and-mqtt-a-simple-guide-cb0cbba1931c (pristupljeno 15. kolovoza 2022.)

[11] "An Introduction to MQTT with NestJS Framework“. https://myas92.medium.com/an-introduction-to-mqtt-with-nestjs-framework-a69ec55483f0 (pristupljeno 26. kolovoza 2022.)

[12] "MQTT vs. HTTP: which one is the best for Iot?”. https://medium.com/mqtt-buddy/mqtt-vs-http-which-one-is-the-best-for-iot-c868169b3105 (pristupljeno 26. kolovoza 2022.)

[13] "MQTT beginner’s guide?”. https://www.u-blox.com/en/blogs/insights/mqtt-beginners-guide (pristupljeno 26. kolovoza 2022.)

[14] "Meet Android Studio“, *Android Developers*. https://developer.android.com/studio/intro (pristupljeno 28. kolovoza 2022.).

[15] Guo, Lin. (2022). “The First Line of Code: Android Programming with Kotlin”. (pristupljeno 28. kolovoza 2022.).

SAŽETAK

Misija i cilj ovog diplomskog rada je bila usporediti rad MQTT i HTTP protokol na istom primjeru kako bi lakše uvidjeli prednosti i nedostaci vezane za slanje i primanje podataka sa servera/uređaja. Korišteni alati su: Android Studio, Kotlin, NestJS, TypeScript, PostgreSQL te u primjerima je korištena implementacija dva protokola: HTTP i MQTT protokola.

Uvidjevši prednosti i nedostatke protokola, navedeno je nekoliko primjera gdje bi pojedini protokol mogao koristiti u razvoju mobilnih/*IoT* aplikacija. Kada govorimo o MQTT i HTTP protokolima, možemo reći da je HTTP jedan od najkorišteniji protokoli za komunikaciju na *World Wide Web-u* (Internetu), a MQTT se zasigurno najviše koristi za povezivanje više uređaja u mrežu zvanu *IoT* (*Internet of Things*). Ali naš cilj je pokazati da se MQTT može koristiti i u mobilnoj aplikaciji i biti svrsishodan, na primjer kao uslugu obavijesti klijenata (poslužitelj šalje obavijesti mobilne uređaje).

Ključne riječi: Android, development, HTTP, IoT, MQTT, protokoli.

ABSTRACT

COMPARISON OF MQTT AND HTTP PROTOCOLS IN THE IOT ENVIRONMENT

The mission and goal of this thesis was to compare the work of MQTT and HTTP protocol on the same example to more easily see the advantages and disadvantages of sending and receiving data from servers / devices. The tools used are: Android Studio, Kotlin, NestJS, TypeScript, PostgreSQL and in implementation two protocols were used: HTTP and MQTT protocols.

Discovering the advantages and disadvantages of the protocol, several examples are given where a particular protocol could be used in the development of mobile / IoT applications. When we are talking about MQTT and HTTP protocols, we can say that HTTP is one of the globally most used protocols for communication on the World Wide Web (Internet), and one is for sure most used for connecting more devices in the networking called IoT (Internet of Things). But our goal is to show that MQTT can be also used in the mobile application and have purpose, for example as notification service (server is notifying mobile devices).

Keywords: Android, development, HTTP, IoT, MQTT, protocols.

ŽIVOTOPIS

Tomislav Kušević rođen je 28.05.1996. u Koprivnici u Republici Hrvatskoj. Nakon završene osnovne škole, 2011. upisuje opću Gimnaziju "Fran Galović" u Koprivnici koju završava 2015. godine. Po završetku srednje škole upisuje Stručni studij Informatike na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Krajem 2017. godine odrađuje praksu kao "Android developer", nakon koje ostaje raditi u firmi COBE u Osijek. Potom 2018. godine završava stručni studij i krajem 2018. prelazi u Code Consulting u Vukovaru, kao *backend developer*. Zatim 2020. godine upisuje diplomski studij Računarstva, smjer Programsko inženjerstvo.

Potpis autora