# Upravljanje IoT uređajima pomoću MQTT poruka

**Paskal, Šimec**

**Master's thesis / Diplomski rad**

**2020**

*Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj:* **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

*Permanent link / Trajna poveznica:* https://urn.nsk.hr/urn:nbn:hr:211:938967

*Rights / Prava:* Attribution-NoDerivs 3.0 Unported/Imenovanje-Bez prerada 3.0

*Download date / Datum preuzimanja:* **2024-03-29**

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

*Repository / Repozitorij:*

Faculty of Organization and Informatics - Digital Repository

zir.nsk.hr

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Paskal Šimec**

# UPRAVLJANJE IOT UREĐAJIMA POMOĆU MQTT PORUKA

**DIPLOMSKI RAD**

**Varaždin, 2020.**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Paskal Šimec**

**Matični broj: 0016120052**

**Studij: Informacijsko i programsko inženjerstvo**

# Upravljanje IoT uređajima pomoću MQTT poruka

**DIPLOMSKI RAD**

**Mentor:**

Prof. dr. sc. Dragutin Kermek

**Varaždin, rujan 2020.**

*Paskal Šimec*

**Izjava o izvornosti**

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

*Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi*

_____

## Sažetak

U ovom radu predstavljen je koncept Internet stvari i načini integracije računalnih uređaja unutar kompleksnog sustava. Definirana je arhitektura Interneta stvari, najčešća primjena kao i vizija koncepta. Osim toga, rad teorijski obrađuje i brojne protokole za upravljanje računalnim uređajima. Glavni protokol kojim se rad bavi je MQTT. Definiran je način rada MQTT protokola, komponente koje sudjeluju u radu te brojne implementacije definiranih komponenti. Različite implementacije MQTT poslužitelja su analizirane i međusobno uspoređene. Osim protokola, u radu su predstavljeni i drugi načini komunikacije između programskih sustava kao što su web servisi i JMS. Kako bi se pokazao način implementacije ranije definiranog koncepta Internet stvari, MQTT protokola i metoda komunikacije, izrađen je programski sustav koji objedinjuje navedene komponente. Izrađeni programski sustav, naziva „Sustav za upravljanjem IoT uređajima“, sastoji se od više povezanih uređaja koji preko MQTT poslužitelja komuniciraju s dvije odvojene web aplikacije. Implementirani sustav prikazuje načine povezivanja uređaja i sustava te različite mogućnosti upravljanja uređajima putem nadzornog sustava. Rad je napravljen u okviru Laboratorija za web arhitekture, tehnologije, servise i sučelja.



**Ključne riječi:** IoT, MQTT, upravljanje web aplikacije, web servisi, REST, SOAP, JMS

# Sadržaj

Sadržaj ..... iii
1. Uvod ..... 1
2. Internet stvari ..... 2
2.1. Arhitektura ..... 3
2.2. IoT protokoli ..... 5
2.3. Primjena ..... 7
2.4. IoT u budućnosti ..... 9
3. MQTT ..... 9
3.1. Princip objavi / pretplati ..... 10
3.2. Uspostava veze između klijenta i poslužitelja ..... 12
3.3. Dodatne funkcije ..... 13
3.3.1. QoS ..... 13
3.3.2. Retain ..... 15
3.3.3. Will ..... 15
3.4. Posrednici ..... 16
3.4.1. Mosquitto ..... 16
3.4.2. ActiveMQ ..... 16
3.4.3. Moquette ..... 16
3.4.4. RabbitMQ ..... 16
3.4.5. Usporedna analiza posrednika ..... 17
4. Upravljanje sustavom ..... 18
4.1. IoT upravljan MQTT porukama ..... 18
4.1.1. Arhitektura IoT sustava upravljanog MQTT porukama ..... 18
4.1.2. Usporedba IoT sustava temeljenog na HTTP i MQTT ..... 20
4.1.3. IoT uređaj kao MQTT posrednik ..... 22
4.2. JMS ..... 23
4.3. Web servisi ..... 25
4.3.1. SOAP ..... 26
4.3.2. REST ..... 29
4.3.3. Usporedba web servisa ..... 32
5. Implementacija sustava ..... 35
5.1. Opis i funkcionalnosti sustava ..... 35
5.2. Korišteni alati i tehnologije ..... 37
5.3. Arhitektura sustava ..... 38
5.3.1. Dijagram slučaja korištenja sustava ..... 38
5.3.2. Dijagram slijeda aktivnosti ..... 39
5.3.3. Dijagram klasa ..... 43

5.3.4. ERA model ..... 45
5.4. Komponente sustava ..... 46
5.4.1. IoT uređaji ..... 46
5.4.2. MQTT poslužitelj ..... 51
5.4.3. Baza podataka ..... 53
5.4.4. Web servisi ..... 54
5.5. Prikaz aplikacije ..... 56
6. Zaključak ..... 63
Popis literature ..... 64
Popis slika ..... 67
Popis tablica ..... 69

# 1. Uvod

U današnjem svijetu interneta sve je češća primjena web aplikacija koje korisnicima pružaju razne korisnike informacije te pružaju sučelje za izvođenje različitih procesa. S velikim povećanjem popularnosti web stranica brzo se razvija popratna tehnologija za izradu stranica, ali i tehnologija koja se koristi uz web stranice kao izvor pružanja drugih usluga. Jedan od vrlo bitnih i popularnih koncepata je korištenje računalnih uređaja u svrhu automatizacije procesa. Takav se koncept popularno naziva Internet stvari. Uvođenjem Interneta stvari brojni poslovi iz svakodnevnog, ali i poslovnog svijeta postali su automatizirani ili praćeni od strane računalnog uređaja. Korištenje računalnih komponenti u životu informatičari smatraju kao korakom u pravom smjeru. Smatra se da će se s godinama taj koncept sve više koristiti te će se područje primjena znatno proširiti.

Kako Internet stvari čine uređaji koji imaju svoju namjenu i u pravilu nisu povezani s web aplikacijama postavlja se pitanje kako kontrolirati i nadzirati takve uređaji. Postoje brojni protokoli koji svojim karakteristikama odgovaraju ovakvom problemu, jedan od njih je MQTT protokol. Korištenjem laganih protokola udaljene računalne uređaje moguće je povezati s odgovarajućim sustavom s ciljem obrade podataka, analize podataka, izvršavanja određenih akcija na temelju obrađenih podataka te u konačnici prikaz podatka krajnjem korisniku.

U ovom radu fokus je stavljen na ranije spomenuti koncept Internet stvari te načinu na koji se može upravljati računalnim uređajima, prikupljati podatke te obrađivati iste. Prikazane su različite alternative povezivanja uređaja i sustava, koje su međusobno uspoređene kako bi se dobio bolji dojam o tome koja je alternativa prikladnija u kojem scenariju..

Rad se može podijeliti u dvije cjeline. Prva cjelina odnosi se na teorijsku osnovu vezanu za Internet stvari i mogućnosti povezivanja. U početku je opisan koncept Internet stvari, arhitektura, način rada te sama primjena. Opisan je MQTT protokol kao jedan od dostupnih protokola za upravljanje IoT uređajima. Na kraju teorijskog dijela opisani su načini komunikacija udaljenih sustava. Druga cjelina rada odnosi se na praktični rad. Praktični rad objedinjuje ranije spomenute koncepte i protokole u potpuno funkcionalan sustav. Sustav se sastoji od uređaja kojima se može upravljati i pratiti njihov rad pomoću nadzorne aplikacije.

# 2. Internet stvari

Pretraživanje pojma "IoT" (engl. *Internet of Things*) pomoću Google-a 2012. godine rezultiralo bi naslovima kao što su "Iluminate of Thanateros" i "International Oceanic Travel Organisation". Dvije godine kasnije, to se drastično promijenilo. U 2014., IoT bio je jedan od ne baš poznatih pojmova u IT industriji [1].

Internet stvari ili popularnije IoT je nova paradigma koja se veoma brzo proširila područjem moderne bežične komunikacije(engl. *wireless*). IoT ima jedinstveni cilj proširiti internet u stvarni svijet obuhvaćajući svakodnevne uređaje u bilo kojem vremenu. Tako fizički uređaji više nisu odvojeni od stvarnog svijeta, već se mogu upravljati daljinski od bilo kuda. Neupitno, glavna konzistentnost IoT ideje je utjecaj na aspekte svakodnevnog života [2].

Povećanjem mogućnosti primjena IoT-a povećavaju se i zahtjevi za razinom sigurnosti koji sustav mora podržavati. Kako bi se osigurala željena razina sigurnosti potrebno je poznavati taksonomiju sigurnosti IoT sustava s obzirom na različite operativne razine. U nastavku slijedi taksonomija koju definira „*A survey of practical security vulnerabilities in real IoT devices*“ [3] .

Informacijska razina sigurnosti trebala bi osigurati sljedeće zahtjeve :

- Integritet: primljeni podaci ne bi se smjeli uređivati tijekom prijenosa.
- Anonimnost: identitet izvora podataka ne bi trebao biti dostupan aplikacijama treće strane.
- Povjerljivost: podaci ne bi smjeli biti dostupni trećoj strani. Potrebno je uspostaviti pouzdan odnos između IoT uređaja.
- Privatnost: korisnički privatni podaci ne bi smjeli biti otkriveni tijekom razmjena podataka.

Razina pristupa određuje sigurnosne mehanizme za kontrolu pristupa mreži kroz funkcionalnosti:

- Kontrolna točka: garantira da samo legitimni korisnici mogu pristupiti IoT uređajima i mreži (npr. reprogramiranje uređaja ili kontrola uređaja putem mreže).
- Autentifikacija: provjerava je li uređaj ima pravo pristupiti mreži i imali li mreža pravo primiti konekciju uređaja.

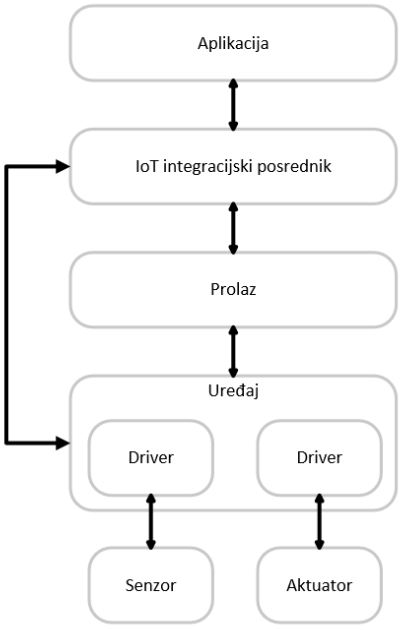
- Autorizacija: osigurava da samo autorizirani uređaji i korisnici imaju omogućen pristup mrežnim uslugama i dostupnim resursima.

Funkcijska razina definira sigurnosne zahtjeve u pogledu sljedećih kriterija:

- Otpornost: odnosi se na razinu mrežnih kapaciteta koji moraju biti osigurani IoT uređajima čak i u slučaju greške u sustavu ili napada na sustav
- Samoorganizacija: označava sposobnost IoT sustava da se prilagodi kako bi ostao operativan čak i u slučaju kvara dijelova ili zlonamjernih napada.

## 2.1. Arhitektura

Arhitektura IoT-a, prikazana na slici 1, uvelike se razlikuje od arhitektura drugih sustava. Glavni elementi koji ovu arhitekturu razlikuju od drugih je mogućnost izostavljanja pojedine komponente arhitekture. Uzmimo za primjer sustav za mjerenje vlage tla: takvom sustavu nije potreban aktuator i može se izostaviti. Iz navedenog primjera dolazimo do zaključka da je arhitektura IoT jednostavna i fleksibilna. Slika 1 prikazuje komponente koje se mogu koristiti prilikom izgradnje IoT sustava. U nastavku su opisane komponente od najnižeg sloja k višem.




Slika 1: Arhitektura IoT sustava [4]

Senzor (engl. *Sensor*) je hardverska komponenta koja služi za mjerenje podataka iz svojeg okruženja te prevođenje prikupljenih podataka u električne signale koji se dalje mogu obrađivati i koristiti u različite svrhe [4].

Aktuator (engl. *Actuator*) je također hardverska komponenta koja ima upravljačku svrhu, a može se koristiti za kontroliranje ili manipuliranje fizičkim okruženjem. Aktuatori rade tako da primaju električne signale od svojih povezanih uređaja te ih prevode u određene fizičke akcije. Poput senzora, aktuatori su obično povezani ili čak integrirani u uređaj, pri čemu vezu mogu uspostaviti žicama ili bežično [4].

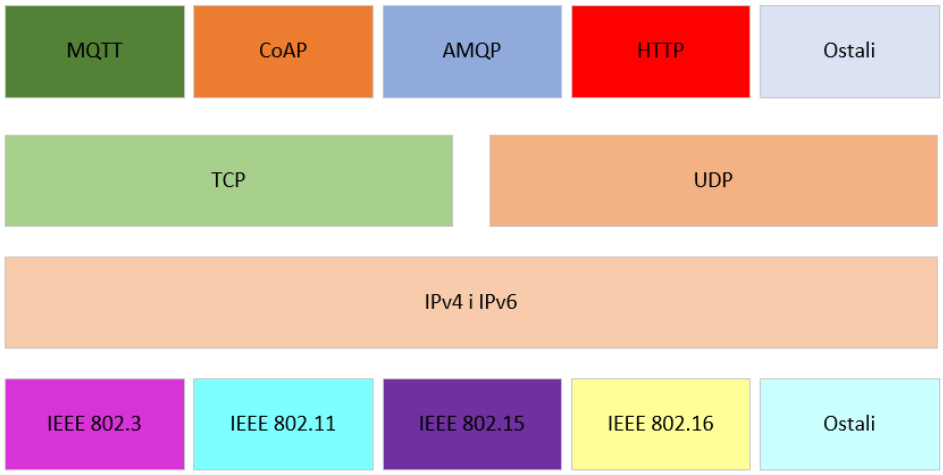
Uređaj (engl. *Device*) je hardverska komponenta koja je izravno povezana na senzore i / ili aktuatore putem žice ili u današnje vrijeme bežično. Kako bi uređaj mogao upravljati aktuatorima ili prikupljati podatke sa senzora sadrži tzv. „Driver“ koji povezuje dvije spomenute komponente. Pomoću drivera omogućuje se slanje signala programskim putem aktuatoru ili obrnuto, prevođenje električni signali dobivenih od senzora u programske vrijednosti. [4]

Prolaz (engl. *Gateway*) se koristi kada se uređaji žele povezati s drugim uređajima ili vanjskim sustavima, a međusobno ne mogu komunicirati određenim protokolom zbog tehničkih ograničenja. Na primjer, uređaj komunicira putem prolaza koristeći IoT protokol, poput ZigBee ili MQTT. Kada prolaz primi poruku u binarnom formatu s uređaja, informacije prevodi u JSON ili XML te ih prosljeđuje sustavu na udaljenoj lokaciji. Isto tako, prolaz može prevesti naredbe u komunikacijske tehnologije, protokole i formate koje podržava odgovarajući uređaj [4].

IoT integracijski posrednik (engl. IoT Integration Middleware) odgovoran je za primanje podatka s povezanih uređaja kao i procesiranje dobivenih podataka. Uređaj može direktno komunicirati s IoT posrednikom ako ima podršku za komunikacijsku tehnologiju kao što je WiFi, Bluetooth ili komunikacijski protokol kao HTTP te podršku za formate prijenosa poruka kao što su JSON i XML. U suprotnom, uređaj komunicira s IoT posrednikom pomoću prolaza.
Aplikacija predstavlja softver koji koristi IoT integracijski posrednik za mogućnost prikupljanja podataka sa senzora ili slanjem signala za upravljanje aktuatorom. [4]

## 2.2. IoT protokoli

U prethodnom poglavlju opisana je arhitektura IoT te pojedine komponente koje su sastavnica arhitekture. Također je objašnjeno je kako se podaci koji preko senzora dođu u uređaj dalje šalju do aplikacije određenim protokolom. Cilj ovog poglavlja je upoznati se s četiri najpoznatija protokola koja se koriste u kombinaciji s IoT-om: MQTT, CoAP, AMQP i HTTP. Navedeni protokoli kao i cijela struktura protokola prikazana je na slici 2.




Slika 2: Prikaz IoT protokola [5]

MQTT (engl. *Message Queuing Telemetry Transport*) jedan je od najstarijih M2M (engl. *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 [5]. Više o MQTT porukama opisano je u nastavku rada.

CoAP (engl. *Constrained Application Protocol*) je web protokol koji radi povrh UDP protokola, a najveću upotrebu ima kod IoT-a sustava. CoAP je varijanta popularnog HTTP, s razlikom da je prilagođena IoT uređajima i M2M komunikaciji. Glavna karakteristika CoAP je njegova jednostavna upotreba, što ga razlikuje od sličnog HTTP [6].

AMQP (engl. *Advanced Message Queuing Protocol*) je protokol aplikacijskog sloja namijenjen za sustave orijentiran na poruke. Zadaća protokola je povezati veliki broj udaljenih aplikacija i sustava s ciljem povećanja interoperabilnosti. Prema arhitekturi, AMQP je vrlo sličan MQTT protokolu. Kao i MQTT sastoji se od poslužitelja, klijenta, objave i preplate na poruke, ali za razliku od MQTT protokola uključuje i mehanizme za razmjenu poruka [3].

Tim Berners-Lee početno je razvio je HTTP (engl. *Hyper Text Transport Protocol*) koji je 1997 godine do kraja razvijen i objavljen od strane IETF i W3C. HTTP svoj na temelji na zahtjev / odgovor arhitekturi . Za razliku od AMQP i MQTT, HTTP koristi univerzalni identifikator resursa (URI) umjesto tema. HTTP je tekstualni protokol i ne određuje veličinu opterećenja zaglavlja i poruke, već ovisi o web poslužitelju ili programskoj tehnologiji. HTTP koristi TCP kao zadani transportni protokol, a TLS / SSL za sigurnost. Globalno prihvaćen standard za internetske poruke koji nudi nekoliko značajki kao što su trajne veze, zahtijevanje cjevovoda i kodiranje podataka [5].

U nastavku slijedi tablica unutar koje su uspoređeni protokoli MQTT, CoAP, AMQP i HTTP.

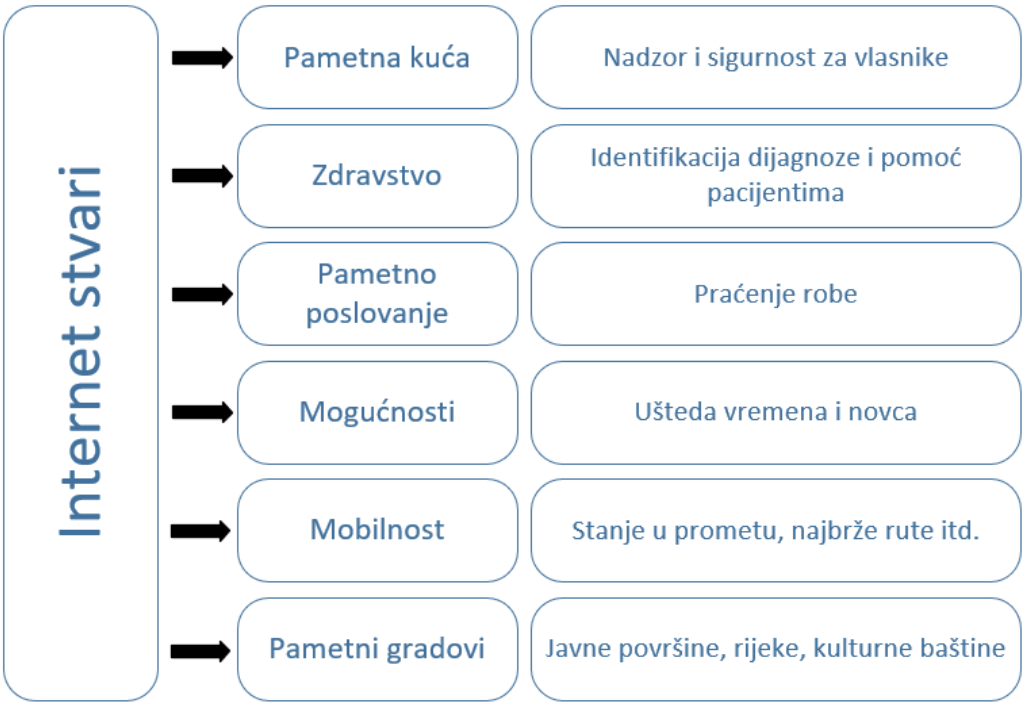
Tablica 1: Analiza IoT protokola [5]

| ***Kriterij*** | **MQTT** | **CoAP** | **AMQP** | **HTTP** |
|---|---|---|---|---|
| *Godina* | 1999 | 2010 | 2003 | 1997 |
| *Arhitektura* | Klijent/Posrednik | Klijent/Poslužitelj<br>ili<br>Klijent/ Posrednik | Klijent/Poslužitelj<br>ili<br>Klijent/ Posrednik | Klijent/Poslužitelj |
| *Način rada* | Objavi/Pretplati | Zatraži/Odgovori<br>Ili<br>Objavi/Pretplati | Zatraži/Odgovori<br>Ili<br>Objavi/Pretplati | Zatraži/Odgovori |
| *Veličina zaglavlja* | 2 Byte | 4 Byte | 8 Byte | Nije definirano |
| *Veličina poruke* | Nije definirano (najviše do 256 MB) | Nije definirano | Nije definirano | Velika |
| *Metode* | Connect, Disconnect, | Get, Post, Put, Delete | Consume, Deliver, Publish, | Get, Post, Head, Put, Patch, |

| | | | | |
|---|---|---|---|---|
| | Publish, Subscribe, Unsubscribe, Close | | Get, Select, Ack, Delete, Nack, Recover, Reject, Open, Close | Options, Connect, Delete |
| *Cache* | Djelomično | Da | Da | Da |
| *Standardi* | TLS/SSL | DTLS, IPSec | TLS/SSL, IPSec, SASL | TLS/SSL |
| *Format kodiranja* | Binary | Binary | Binary | Text |
| *Dostupnost* | Otvorenog koda | Otvorenog koda | Otvorenog koda | Besplatan |

## 2.3. Primjena

IoT nalazi primjenu u različitim područjima od medicine preko prometa do različitih kućnih automatizacija. Također, područje primjene se stalno širi i ljudi uviđaju kako određene poslove mogu pojednostaviti, automatizirati i ubrzati uvođenjem određene komponente IoT-a. Mnogi znanstveni radovi podijelili su primjenu IoT-a u šest odvojenih kategorija prikazanih na slici 3. U nastavku slijedi kratak opis svake navedene kategorije.




Slika 3: Područja primjene IoT-a [2]

Pametna kuća (engl. *Smart Home*) je kuća koja sadrži komunikacijsku mrežu koja povezuje električne uređaje i usluge čime omogućava daljinsko upravljanje, nadzor i pristup. Ključni element pametne kuće je prolaz koji povezuje unutarnju mrežu kuće s internetom. Agent kontrolira Svaki resurs koji se nalazi u kući, primjerice, TV. Agent je proizvođač, potrošač i poslužitelj usluga koje prosljeđuju IoT uređaji [7].

Konstantan i brz razvoj IoT tehnologije omogućuju unaprjeđenje pametnih sustava za podršku i poboljšanje zdravstva (engl. *Healthcare*), kao i poslova u biomedicini. Automatska identifikacija i praćenje ljudi, biomedicinske uređaje u bolnicama, ispravno određivanje doze lijekova i praćenje pacijenata u stvarnom vremenu, parametri za rano otkrivanje kliničkog pogoršanja samo su neke od mogućih primjena IoT-a u zdravstvenom sektoru [8].

Unutar poslovnog svijeta često se spominju tzv. pametni poslovi (engl. *Smart Business*) koji svoje poslovanje temelje na IoT tehnologiji, točnije na RFID. RFID (engl. *Radio-frequency identification*) tehnologija koristi se kod upravljanje zalihama, lancem nabave i isporukama pri čemu se oslanja na mogućnost RFID za prepoznavanje i podrške u praćenju resursa. Svaki proizvod sadrži RFID oznaku koja se čita pomoću posebnih RFID čitača te šalje u sustav kako bi se mogla analizirati i izvršiti potrebne akcije [2].

U svakodnevnom, ali i poslovnom životu korištenje različitih IoT usluga (engl. *Utilities*) može uštedjeti vrijeme i novac. Primjerice, jedna od najvažnijih mogućnosti IoT-a je nadgledanje vode za piće. Senzori koji mjere vanjske parametre kao što su vlaga i temperature, instaliraju se na potrebnom lokacijama kako bi se osigurala kvaliteta opskrbe vodom. Ista mreža može se u koristiti u poljoprivredi za uštedu novca i vremena nadgledanjem trave na daljinu. Nadgledanjem vlažnosti i kvalitete zraka mogu se spriječiti onečišćenja i izbjeći nalijevanje uz pomoć IoT-a [2].

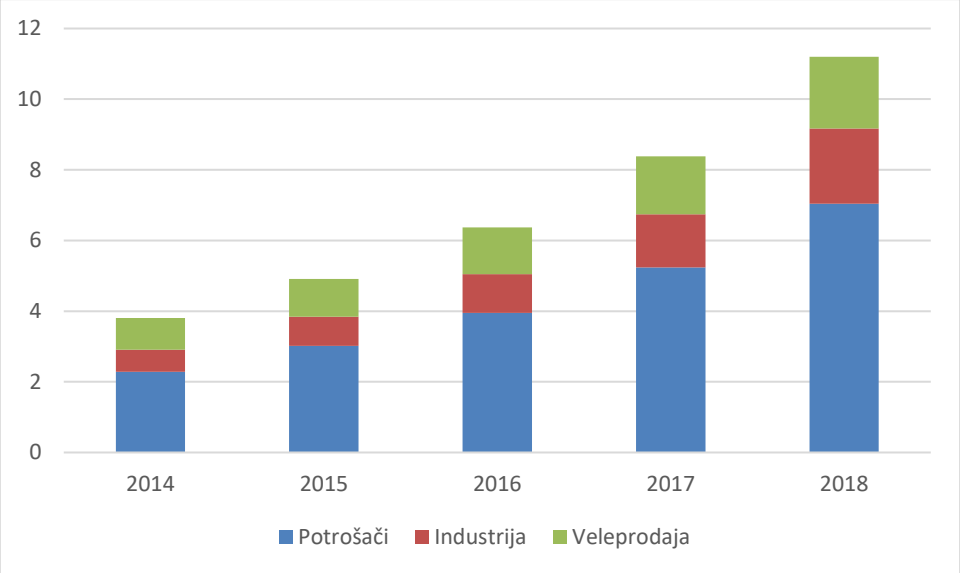
Cestovni promet je još jedno područje u kojem IoT ima veliku ulogu. Korištenjem IoT tehnologije omogućuje se mrežno praćenje putovanja tako da IoT može birati rutu, red prometa, zagađenje zraka i emisije buke. IoT može promijeniti algoritam gradskog prometa koristeći mobilnu komunikaciju i označavanjem cesta kao internetskih usluge s ciljem povećanja kvalitete usluga u gradu [2].

Pametni gradovi (engl. *Smart Cities*) su koncept u kojem bi se IoT koristio s ciljem uvođenja ICT u upravljanju javnim poslovima. Koncept pametnih gradova temelji se na cilju boljeg korištenja javnih resursa, povećanju kvalitete usluga koje se nude građanima, uz smanjenje smanjenja operativni troškovi javnih uprava. IoT doista može donijeti niz koristi u

upravljanju i optimizaciji tradicionalnih javnih usluga, kao što su prijevoz i parkiranje, rasvjeta, nadzor i održavanje javnih površina, očuvanje kulturne baštine, prikupljanje smeća, kontrola bolnica i škola [9].

## 2.4. IoT u budućnosti

Od uvođenja IoT tehnologija, potražnja i korištenje sustava koji se temelje na IoT-u znatno su porasli. Svakim danom pokreću se novi projekti koji se temelje na IoT-u, razvijaju se nove tehnologije koje uključuju IoT, a u stare se integriraju komponente IoT kako bi aplikacije ostale ažurne i mogle pratiti najnoviju tehnologiju. U nastavku slijedi graf koji predstavlja rast područja primjene IoT-a u periodu od pet godina kao i područja njegove primjene. Iz slike možemo uočiti trend rasta korištenja IoT-a i možemo vidjeti da IoT postiže najveći porast u korištenju kod potrošača, zatim u industriji, a najmanje se koristi u veleprodaji.




Slika 4: Rast i područje primjene IoT-a [10]

Bitno je napomenuti da se i s razvojem i uvođenjem 5G mreže otvaraju nove domene aplikacija unutar IoT-a. Uvođenje 5G mreže rezultirat će mnoštvom novih mogućnosti vezanih za IoT te potencijalne promjene u načinu rada IoT-a, novih protokola i unapređenje IoT arhitekture [11].

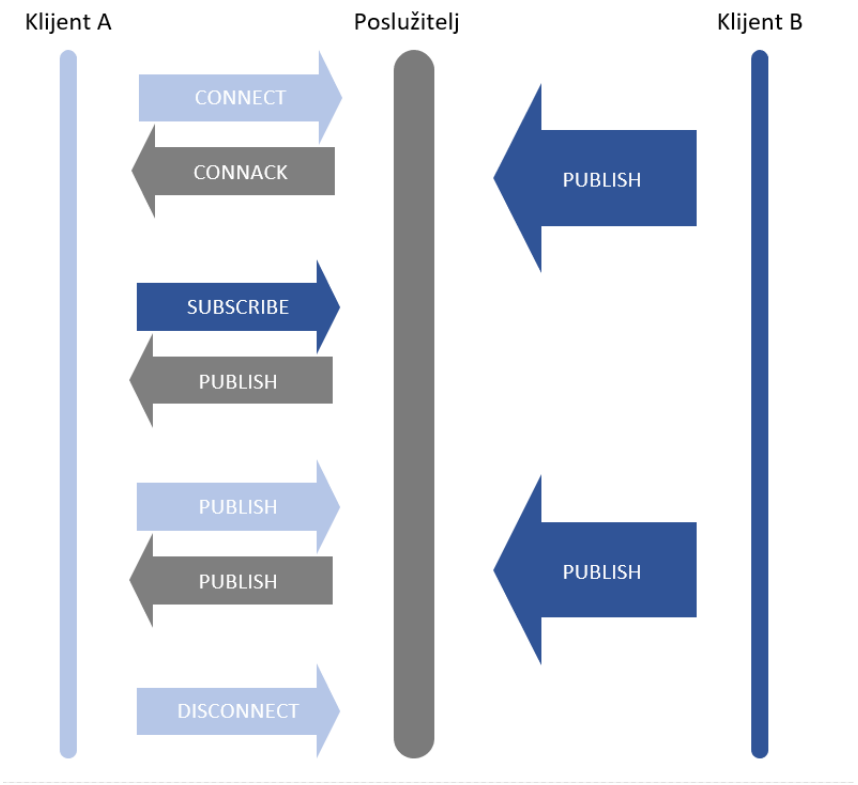
# 3. MQTT

MQTT (engl. *Message Queuing Telemetry Transport*) je klijentski poslužitelj koji radi na principu objavi / pretplati (engl. *Publish / subscribe*), a koristi se za prijenos poruka. Radi se o jednostavnom, laganom protokolu otvorenog koda koji je namijenjen za jednostavnu upotrebu. Činjenica da se radi o laganom i jednostavnom protokolu čini MQTT idealnim za upotrebu u mnogim situacijama poput komunikacije u kontekstima M2M i IoT [12].

## 3.1. Princip objavi / pretplati

Kao što je u uvodnom dijelu napisano, MQTT temelji se na principu objavi / pretplati. Svaki protokol s jedne strane ima klijenta pa tako i MQTT. Klijent koji objavljuje poruke, također poznat kao objavljivač, je potpuno odvojen od klijenta koji prima poruke, radi se o potpuno neovisnoj vezi u kojoj jedan klijent nema nikakva saznanja o ostalim klijentima. Klijent može objaviti poruke određene vrste i samo će klijenti koje zanimaju određene vrste poruka primiti objavljene poruke.

Objavi/pretplati princip zahtjeva i poslužitelja koji se u kontekstu MQTT-a često naziv *Broker*. Svaki klijent koji želi poslati poruke prethodno mora uspostaviti vezu s poslužiteljem. Nakon što klijent pošalje poruke, poslužitelj ih filtrira i distribuira onim klijentima koji su zainteresirani za tu vrstu poruka. Naravno, da bi klijent dobio određenu poruku, prethodno se kod poslužitelja mora registrirati kao zainteresiran za određene vrste poruka. Takvi registrirani klijenti nazivaju se pretplatnici. Kako opisani tok poruka bio pravilno uspostavljen, objavljivač i pretplatnik moraju biti registrirani na istu vrstu poruka kod poslužitelja [13]. Opisani proces razmjene poruke prikazan je na slici broj 5**.**




Slika 5: Proces razmjene poruka pomoću MQTT protokola

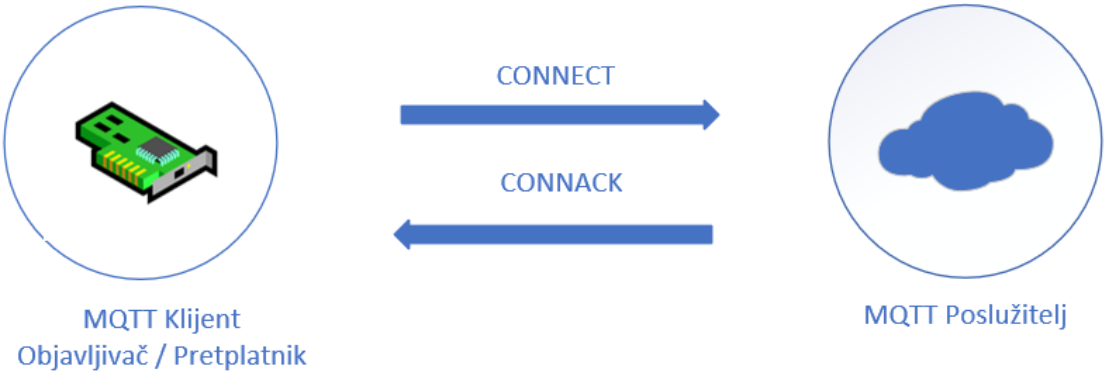
Objavljivači i pretplatnici su potpuno odvojeni jedni od drugih i nemaju nikakvu poveznicu, a ne moraju biti aktivni u isto vrijeme. Postoji scenarij u kojem objavljivač objavi poruku, a pretplatnik ju primi kasnije. Proces objave poruke nije sinkroni proces u odnosu na primanje poruka. Objavljivač zahtjeva od poslužitelja da objavi poruku, dok s druge strane klijenti koji su pretplaćeni na te poruke, poruke mogu primiti u različita vremena. Ovim principom izbjegava se blokiranje objavljivača dok pretplatnik ne primi poruku. Također je moguće poslati poruku poslužitelju kao sinkroni proces prema poslužitelju i nastaviti izvršenje posla tek nakon što je proces uspješno završen. U većini slučajeva preferira se asinkroni način rada jer smanjuje ovisnost među sudionicima procesa [13].

Pozitivna karakteristika MQTT-a je ta da objavljivač može poslati poruku stotinama klijenata slanjem samo jedne poruke prema poslužitelju. Nakon slanja jedne poruke poslužitelj je odgovoran za slanje poruke svim klijentima koji su se pretplatili na odgovarajuću temu. Budući da su objavljivači i pretplatnici razdvojeni, objavljivač ne zna postoji li pretplatnik koji će primiti poslanu poruku. Zbog toga je ponekad potrebno da pretplatnik postane i objavljivač te objavi i primi poruku te time potvrdi uspješnost izvršenja procesa [13].

## 3.2. Uspostava veze između klijenta i poslužitelja

MQTT poslužitelj je središte MQTT protokola te samim time i glavna jedinica u objavi / pretplati principu. Jedna od zadaća poslužitelja je provjera autentičnosti i autorizacija klijenata koji će objavljivati poruke, odnosno pretplatiti se na njih. Prema tome, prva i najbitnija relacija između klijenta i poslužitelja je uspostava početne veze.

Kako bi se veza uspješno uspostavila, klijent mora poslati CONNECT poruku poslužitelju koja sadrži sve potrebne podatke za inicijalizaciju veze i indiciranje autentifikacije i autorizacije klijenta. MQTT poslužitelj provjerava CONNECT poruku, izvršava autentifikaciju i autorizaciju klijenta te klijentu šalje odgovor naziv CONNACK. U slučaju da klijent pošalje nevažeću CONNECT poruku, poslužitelj automatski zatvara vezu [13]. Opisani proces prikazan je na slici 6.




Slika 6: Uspostava veze između klijenta i poslužitelja [13]

## 3.3. Dodatne funkcije

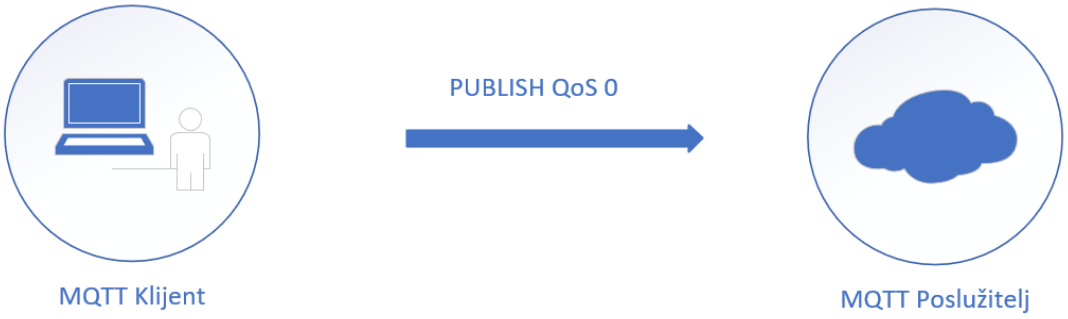
Unutar ovog poglavlja opisane su tri glavne funkcije unutar MQTT protokola koje su dostupne klijentima: *QoS*, *Retain* i *Will*.

### 3.3.1. QoS

*QoS* (engl. *Quality of Service*) pruža mogućnost konfiguriranja razine informacija o isporuci poruka. Klijent i poslužitelj potvrđuju isporuku poruke i ponovno ju šalju ako je to potrebno [14].

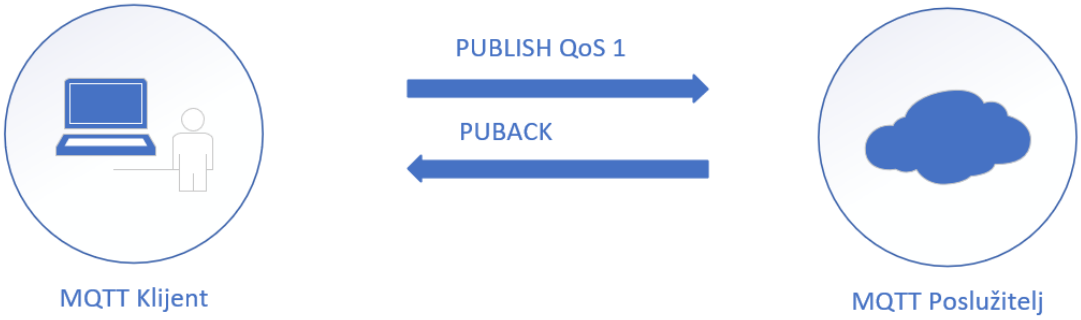
Postoji tri razine QoS:

1) Najviše jedna isporuka (engl. *At most once delivery*) je nulta razina QoS. Ova razina usluge garantira najbolju isporuku. Primatelj ne potvrđuje primanje poruke, dok pošiljatelj ne pohranjuje i ne šalje ponovno poruku u slučaju da ona nije došla do primatelja. Popularan naziv nulte razine je *Pošalji i zaboravi* [15].



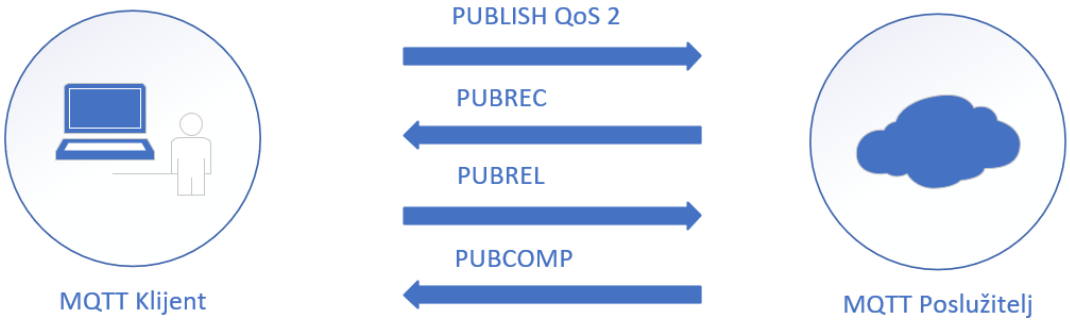

Slika 7: Prikaz QoS 0 razine

2) Prva razina QoS naziva Barem jedna isporuka (engl. *At least once delivery*) kao što joj i ime govori, jamči barem jednu dostavu poruke primatelju. Pošiljatelj pohranjuje poruku sve dok od primatelja ne dobije PUBACK paket koji potvrđuje primanje poruke. Moguće je da se jedna poruka više puta šalje [15].




Slika 8: Prikaz QoS 1 razine

3) Posljednja razina QoS naziva Točno jedna isporuka (engl. *Exactly once delivery*), jamči da će svaka poruka poslana od strane primatelja bude točno jednom dostavljena. Navedena karakteristika QoS razine čini ju najsigurnijom ali ujedno i najsporijom od tri dostupne razine. Sigurnost isporuke poruke je realizirana kroz minimalno dva zahtjev/odgovor toka 15]




Slika 9: Prikaz QoS 2 razine

U studiji pod nazivom „*Correlation Analysis of MQTT Loss and Delay According to QoS Level*“ [16] napravljena je analiza brzine prijenosa podataka prilikom korištenja različitih QoS konfiguracija kod žičanog i bežičnog interneta, a dobiveni rezultati prikazani su u tablici 2. Iz tablice je vidljivo da povećanjem razine QoS raste i vrijeme prijenosa. Autori rada zaključuju da postoji velika ovisnost QoS razine i vremena prijenosa podataka [16].

Tablica 2: Analiza brzine prijenos podataka kod različitih QoS konfiguracija [16]

| | **Žičani Internet (u ms)** | **Bežičani Internet (u ms)** |
|---|---|---|
| *QoS 0* | 0.771574 | 0.878897 |
| *QoS 1* | 0.788814 | 0.884428 |
| *QoS 2* | 0.991416 | 0.904699 |

### 3.3.2. Retain

*Retain* pruža mogućnost slanja zadnje poslane poruke novom pretplatniku. Za ovu funkciju postoji zastavica naziva *Retain*. Ako je postavljena zastavica, poslužitelj interno pohranjuje poruku sve kod ne dođe nova poruka koja također ima postavljenu zastavicu. Pohranjena poruka je proslijeđena novom pretplatniku teme [14].

### 3.3.3. Will

*Will* pruža mogućnost informiranja o neočekivanom prekidu veze. Za ovu funkciju postoji zastavica naziva *Will* koja se postavlja prilikom ustave veze s poslužiteljem, a zastavica se sastoji od poruke. Ako je postavljena zastavica, poslužitelj sprema *Will* poruku unutar interne memorije prilikom uspostave veze s klijentom. U slučaju da poslužitelj dobije informaciju o zatvaranju veze prema klijentu, *Will* poruka bit će proslijeđena do klijenta [14].

## 3.4. Posrednici

MQTT poslužitelj ili posrednik glavna je komponenta MQTT protokola. Brojne implementacije tog poslužitelja su javno dostupne i jednostavne za korištenje. Unutar ovog poglavlja opisani su neki od popularnijih MQTT poslužitelja te je napravljena usporedna analiza poslužitelja.

### 3.4.1. Mosquitto

Eclipse Mosquitto je MQTT poslužitelj otvorenog koda koji implementira MQTT protokol. Mosquitto je lagan i pogodan za upotrebu na svim uređajima, od jednokratnih računala male snage do potpunih poslužitelja [17]. Često se upotrebljava kao poslužitelj unutar IoT sustava instaliran na IoT uređaju kao što je Raspberry Pi.

### 3.4.2. ActiveMQ

ActiveMQ je najpopularniji poslužitelj otvorenog koda koji pruža podršku za različite protokole. Poslužitelj podržava različite metode integracije kroz programske jezike C, C++ i Python. Najpoznatiji i najčešće korišteni protokoli podržani od strane ActiveMQ su JMS, STOMP i MQTT [18].

### 3.4.3. Moquette

Moquette je implementacija MQTT posrednika napisana u Java programskom jeziku. Glavna karakteristika Moquette poslužitelja je jednostavan programski kod malog opsega. Zbog navedenih karakteristika poslužitelj radi brzo te nema mogućnost paralelnog izvršavanja što u određenim situacijama može izazvati dodatne komplikacije [19].

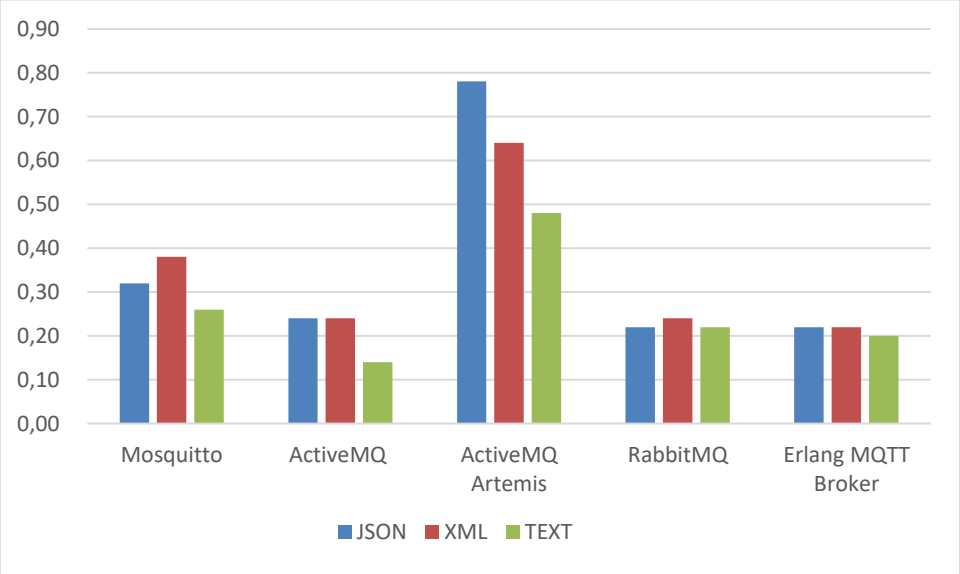
### 3.4.4. RabbitMQ

RabbitMQ je najraširenija implementacija posrednika za upravljanje porukama s otvorenim programskim kodom. Radi se o laganom poslužitelju, jednostavnom za upotrebu s mogućnosti korištenja u oblaku. Glavna odlika RabbitMQ poslužitelja je mogućnost podrške za više odvojenih protokola koji se temelje na porukama [20].

### 3.4.5. Usporedna analiza posrednika

Nakon predstavljanja nekih javno dostupnih posrednika s podrškom za MQTT protokol, unutar ovog poglavlja predstavljeni su rezultati komparativne analize poslužitelja. Analizom se željelo ispitati vrijeme (u milisekundama) od slanja poruke poslužitelju pa do primanja iste poruke.. Poslužitelji na kojima se provodilo istraživanje su: Mosquitto, ActiveMQ, ActiveMQ Artemis, RabbitMQ i Erlang MQTT posrednik. Navedeni posrednici instalirani su na istom računalu s Windows operacijskim sustavom.

Analiza je provedena tako da su svakom od navedenih posrednika poslane tri odvojene poruke uz mjerenje potrebnog vremena. Prva poruka je tipa JSON veličine 238 bajtova, druga tipa XML veličine 360 bajtova, i zadnja poruka tekstualnog tipa veličina 1124 bajtova. Slika 10 prikazuje obrađene i grupirane podatke dobivene analizom.




Slika 10: Rezultati usporedne analize posrednika (autorski rad)

Iz dobivenih rezultata vidljivo je da svi poslužitelji imaju slična vremena primanja poruka. Može se izdvojiti ActiveMQ Artemis poslužitelj kao poslužitelj s ukupno najlošijim rezultatima i ActiveMQ kao poslužitelj s najboljim rezultatima. Također, vidljiva je ovisnost vremena i tipa podataka, odnosno strukture datoteke, a ne same datoteke. Najveća datoteka tekstualnog tipa bila je kod svakog poslužitelja najkraće vrijeme u odnosu na XML i JSON datoteke koje su znatno manje veličine.

# 4. Upravljanje sustavom

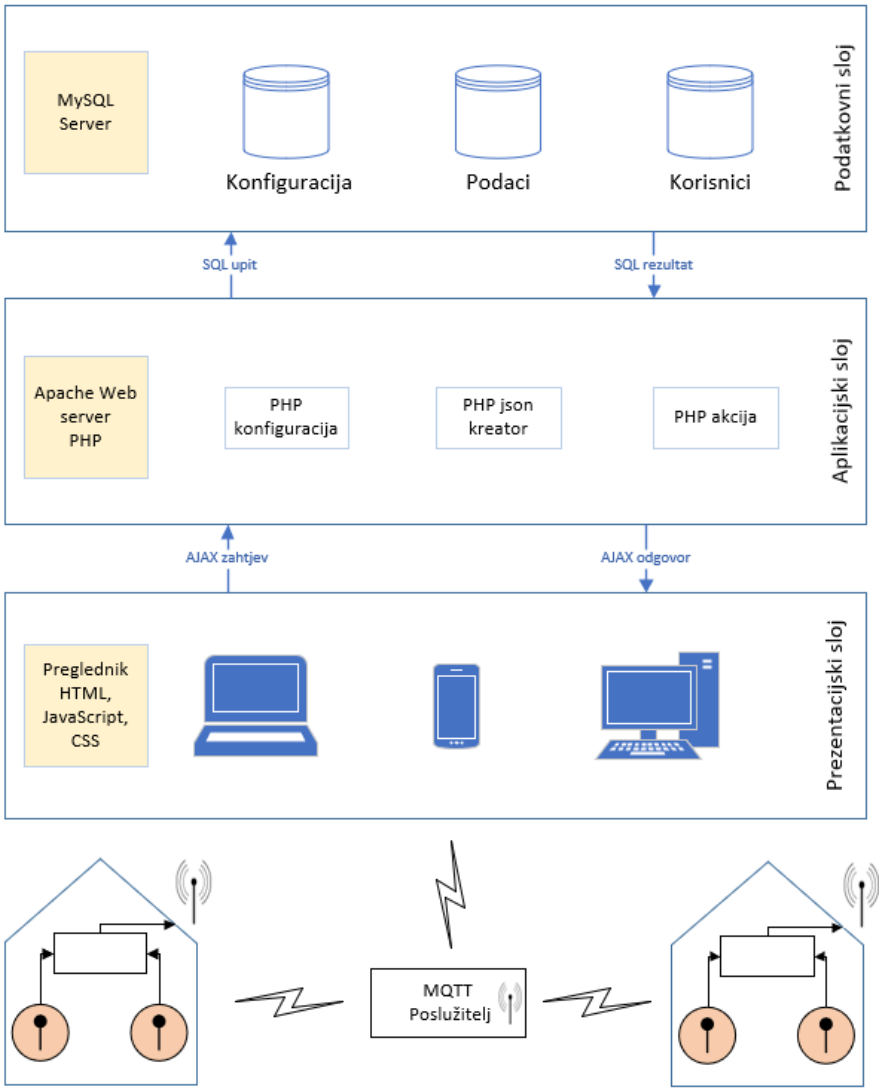
## 4.1. IoT upravljan MQTT porukama

Korištenje MQTT poruka kao poveznica između IoT uređaja u aplikacije je često i preporučljivo rješenje. Razlog tome jest izrazito mala veličina poruka i velika brzina prijenosa. U nastavku slijedi opis arhitekture IoT sustava temeljenog na MQTT porukama te mogućnosti korištenja IoT uređaja kao MQTT posrednika.

### 4.1.1. Arhitektura IoT sustava upravljanog MQTT porukama

Arhitekturu većine web aplikacija moguće je podijeliti u tri sloja: prezentacijski, aplikacijski i podatkovni sloj. Prezentacijski sloj predstavlja korisničko sučelje putem kojeg su korisniku prikazane sve bitne informacije te dostupne akcije. Drugi sloj, naziva aplikacijski sloj, predstavlja složenu logiku koja se nalazi iza web aplikacije. Podatkovni sloj, ujedno i zadnji, sadrži sve potrebne podatke za rad aplikacije kao i mogućnosti upravljanja podacima [21]. Ovakva podjela aplikacija smanjuje međusobnu povezanost komponenti unutar sustava.

Slika 11 predstavlja opisanu arhitekturu, odnosno jednu od mogućih varijacija arhitekture. Sama arhitektura definirana je slojevima, funkcionalnostima i poveznicama, dok su tehnologija, metode komunikacije, podaci i način upravljanja proizvoljni i kao takvi se razlikuju od sustava do sustava. Navedena slika predstavlja arhitekturu sustava koji podatke dobivene od IoT uređaja pomoću MQTT poruka, obrađuje i u konačnici prikazuje korisniku.




Slika 11: Arhitektura sustava temeljenog na MQTT porukama [21]

Prezentacijski sloj temelji se na HTML-u i CSS-u pomoću kojih su podaci oblikovani i prikazani korisniku, dok se JavaScript koristi za dodavanje dinamičnosti aplikaciji. Korištenjem AJAX-a (engl. *Asynchronous JavaScript and XML*) omogućuje se komunikacija prema aplikacijskom sloju. Glavna karakteristika AJAX-a je mogućnost komunikacija prema poslužitelju s ciljem dostavljanja podataka ili izvršavanja zadataka te prikaz rezultata korisniku bez ponovnog učitavanja cijele aplikacije, već samo njezinih elemenata koji sadrže promjene.

IoT uređaji, nakon što očitaju potrebne podatke, šalju MQTT poruku prema MQTT posredniku koji dalje distribuira poruke zainteresiranim klijentima. U konkretnom primjeru, očitani podaci šalju se u aplikacijski sloj gdje je pohranjena sva logika. Aplikacijski sloj obrađuje podatke i prema potrebi ih pohranjuje u bazu podatka pomoću podatkovnog sloja.

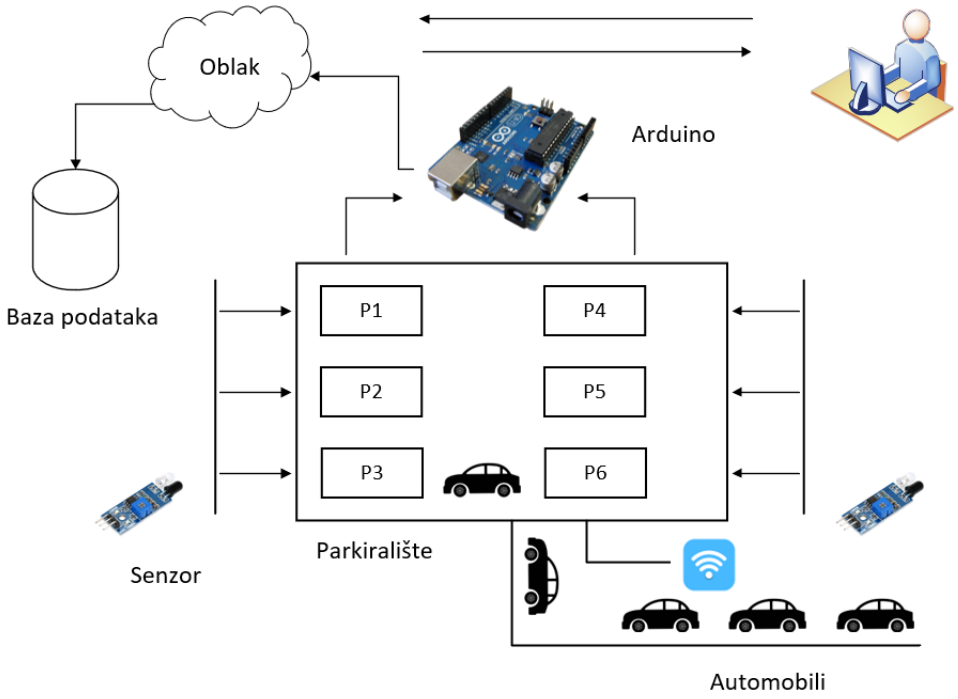
Podatkovni sloj predstavlja vezu prema bazi podataka koja sadrži sve bitne informacije za rad sustava. Jedna skupina pohranjenih podataka predstavljaju podaci očitane s IoT senzora koji se dalje mogu koristiti za analizu i izradu različitih vrsta izvještaja. Podatkovni sloj omogućuje prikaz ne samo trenutno očitanih podatka s IoT senzora, već i prikaz u određenom proteklom trenutku [21].

### 4.1.2. Usporedba IoT sustava temeljenog na HTTP i MQTT

U poglavlju vezanom za IoT navedeni su različiti protokoli koji se mogu koristiti unutar složenog IoT sustava. Unutar ovog poglavlja prikazan je sustav za pametno parkiranje automobila koji koristi neke od ranije definiranih protokola.

U istraživanju naziva „*Intelligent Parking Cloud Services based on IoT using MQTT Protocol*“ [21] predstavljen je sustav za pametno parkiranje automobila korištenjem IoT-a i HTTP. Sustav je zamišljen tako da svaki automobil sadrži *Bluetooth* ili infracrveni uređaj, a parkiralište odnosno svako parkirno mjesto ima infracrveni uređaj koji se spojen na WiFi kako bi mogao ostvariti vezu prema internetu.

Slika 12 prikazuje opisani sustav koji se sastoji od Arduino uređaja kao glavne IoT komponente koja sadrži svu logiku i ostvaruje vezu prema aplikaciji i bazi podataka. Automobili i parkirna mjesta sadrže senzore za razmjenu podataka pomoću infracrvenih zraka. Podaci prikupljeni sa senzora šalju se prema Arduino uređaju koji ih obrađuje i prosljeđuje. Sam korisnik sustava ima uvid u aplikaciju koja grafički prikazuje ranije prikupljene i obrađene podatke o slobodnim, odnosno zauzetim parkirnim mjestima.




Slika 12: Arhitektura sustava za pametno parkiranje automobila [22]

U literaturi se navodi [21], da je nedostatak ovakvog sustava je rad pomoću HTTP. Korištenjem HTTP nije moguće postići željenu razinu skalabilnosti, performansi i kvalitete usluga. Jedan od problema prilikom korištenja HTTP i IoT uređaja je velika potrošnja struje od strane IoT uređaja koja nije poželjna. Također, autori rada navode probleme u vremenskom kašnjenju odgovora.

Kako bi se sustav unaprijedio, integrirana je podrška za MQTT umjesto dosadašnjeg HTTP. Po završetku implementacije MQTT protokola, napravljena je usporedba rada sustava korištenjem HTTP i MQTT protokola. Temeljem analize dobiveni su zaključci [22]:

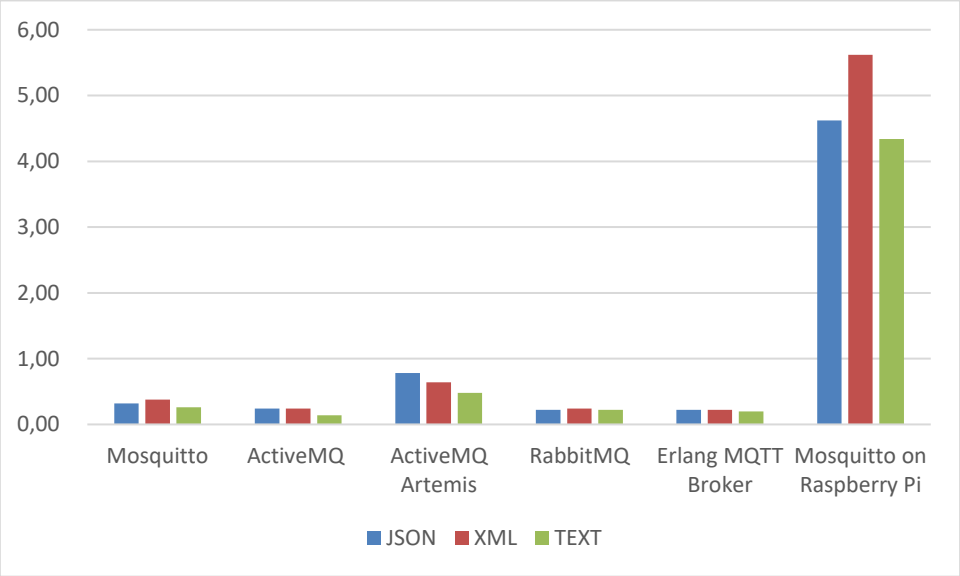
- MQTT poruke su bitno manje veličine od HTTP paketa
- Potrošnja struje je manja kod MQTT protokola i ne raste linearno s povećanjem veličine poruke kao što je slučaj kod HTTP
- Vrijeme odaziva kod MQTT poruka je do 50% kraće u odnosu trajanje kod HTTP

U zaključku [21] autori navode kako predstavljeni sustav pomaže u izbjegavanju sve većih prometnih problema na velikim parkiralištima poput trgovačkih centara i drugih užurbanih područja. Korištenjem MQTT protokola u predstavljenom sustavu drastično će porasti brzina sustava te njegova pouzdanost što će direktno utjecati na smanjenje prometnih problema te gužvi na cestama.

### 4.1.3. IoT uređaj kao MQTT posrednik

Do sada je MQTT posrednik predstavljen kao komponenta MQTT protokola koja je instalirana na udaljenom računalu koji sadrži Windows, Unix ili neki od drugih operacijski sustava. Za razliku od navedenih rješenja, postoji posve drugačiji način implementacije MQTT posrednik pomoću IoT uređaja. IoT uređaji primarno imaju ulogu prikupljanja podataka sa senzora ili upravljanja aktuatorima, ali postoji mogućnost implementacije samog MQTT posrednika na IoT uređaju. Naravno, nemaju svi IoT uređaji podršku za instalaciju posrednika. Jedan od vrlo popularnih IoT uređaja s dobrom podrškom jest Raspberry Pi.

Podacima prethodno izrađene usporedne analize posrednika dodane su vrijednosti MQTT posrednika instaliranog na IoT uređaju. Točnije, radi se o Raspberry PI Zero W uređaju i Mosquitto implementaciji posrednika. Dobiveni rezultati prikazani su na slici 13.




Slika 13: Nadopunjeni rezultati usporedne analize posrednika (autorski rad)

Kao što je i vidljivo iz slike, MQTT posrednik instaliran na IoT uređaju ostvario je daleko lošije rezultate u usporedbi s poslužiteljima na Windows računalu. Jedan od razloga ovako loših rezultata jest mala snaga Raspberry PI uređaja u odnosu na stolno računalo s Windows operacijskim sustavom. Iz dobivenih rezultata može se zaključiti da ako se radi o sustavu koji ima na raspolaganju velike resurse te mora ostvariti visoku razinu pouzdanosti i brzine, ne savjetuje se korištenje IoT uređaja kao MQTT posrednika. S druge strane, ako se radi o malom projektu koji nema produkcijsku namjenu te se žele pokazati mogućnosti IoT uređaja i smanjiti korištenje skupih resursa, poslužitelj instaliran na IoT uređaju je odlično rješenje.

## 4.2. JMS

JMS (engl. *Java Message Service*) API omogućuje aplikacijama da kreiraju, šalju i čitaju poruke. Također, definira zajednički skup sučelja i semantike koji omogućava programima pisanim u programskom jeziku Java komunikaciju s drugim sustavima temeljenim na porukama. JMS minimizira skup pojmova koji programer treba naučiti kako bi koristio proizvode za razmjenu poruka, ali pruža dovoljno mogućnosti za podršku sofisticiranih aplikacija za razmjenu poruka [23].

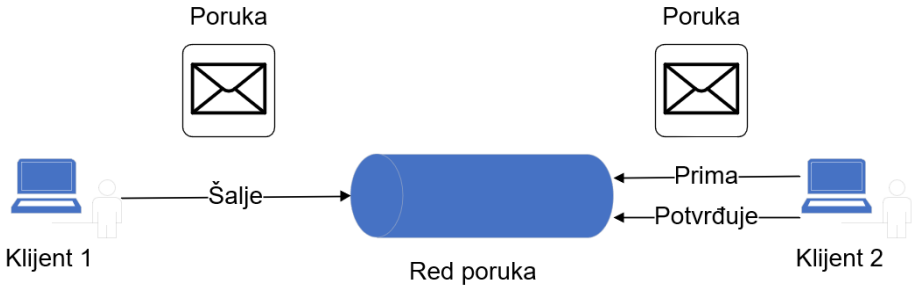
JMS je sadržan unutar JEE (engl. *Java Platform Enterprise Edition*), koja omogućuje stvaranje rješenja za razvoj, upotrebu i upravljanje višeslojnim aplikacijama usmjerenim na poslužitelje. Zadnjim unapređenjima JEE mijenja ime u *Jakarta EE* dok JMS dobiva novi naziv *Jakarta Messaging*. Usprkos promijeni imena brojna literatura se referencira na JMS prema starom nazivu tako da će se stari naziv koristiti i u ovom radu.
JEE koristi Java platformu, standardno izdanje za širenje cjelovite, skalabilne, stabilne, sigurne i brze Java platforme na razini poduzeća. Sama činjenica da je JMS dio JEE znači da se aplikacije koje koriste JMS mogu izvoditi na bilo kojoj platformi koja ima Java virtualni stroj što otvara šire mogućnosti za aplikacije [24].

Prema službenoj dokumentaciji [23], JMS aplikacija svoju arhitekturu temelji na sljedećim komponentama:

- JMS poslužitelj – sustav za razmjenu poruka koji implementira JMS sučelja i pruža administrativne i kontrolne osobine
- JMS klijent – su programi ili komponente, napisane u programskom jeziku Java, koje primaju i šalju poruke
- Poruke – objekti koji komuniciraju informacijama između JMS klijenata
- JMS objekti ili anotacije- služe za bolju integraciju i implementaciju JMS destinacija i veza prema klijentima

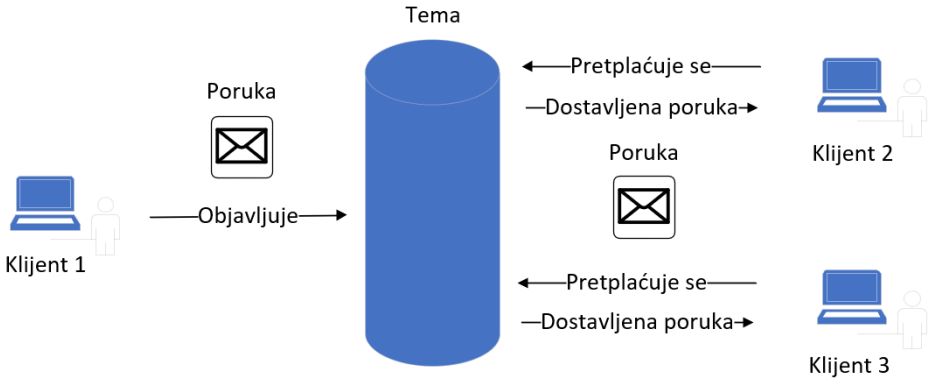
JMS API podržava dva različita stila temeljena na slanju i primanju poruka: PTP (engl. *point-to-point*) i Objavi/Pretplati stil. Glavna karakteristika JMS-a je mogućnost korištenja istog programskog koda neovisno o stilu slanja i primanja poruka. Ta mogućnost aplikacijama osigurava dodatnu fleksibilnost i višekratno korištenje [23].

Stil PTP temelji se na pošiljatelju, primatelju te na redu poruka. Svaka poruka adresirana je na specifičan red, a klijenti koji primaju poruke izdvajaju poruke iz željenog reda. Red čuva primljene poruke sve dok poruke nisu dostavljene primatelju ili ne isteknu. Glavne osobine PTP stila su da svaka poruka ima samo jednog primatelja, dok primatelj može dohvatiti poruku bez obzira je li bi aktivan kada je pošiljatelj poslao poruku ili nije [23]. Slika 14 prikazuje arhitekturu PTP stila.




Slika 14: P2P stil [23]

Drugi podržani stil je već ranije objašnjeni Objavi/Pretplati stil. Arhitektura Objavi/Pretplati stila nalazi se na slici 15.




Slika 15: Objavi / Pretplati stil [23]

Glavne osobine Objavi/Pretplati stila su da svaka poruka može imati više primatelja, dok klijent koji se pretplati na temu može konzumirati samo one poruke koje su poslane nakon što se klijent pretplatio te mora nastaviti biti aktivan kako bi mogao konzumirati poruke

## 4.3. Web servisi

Cilj web servisa jest omogućiti distribuirano okruženje u kojem bilo koji broj aplikacija ili komponenata aplikacije može međusobno djelovati u okruženju koje nije ograničeno platformom ili programskim jezikom. Prema definiciji, web servis je dio poslovne logike lociran na internetu s mogućnosti pristupa pomoću standardnih internet protokola kao što su HTTP ili SMTP [25].

Prema izvoru [25], glave osobine web servisa su:

- XML baziran – korištenjem XML-a kao sloj za predstavljanje podataka eliminira se ovisnost prema internetu, operacijskom sustavu ili određenoj platformi
- Slaba povezanost – sučelje web servisa može se s vremenom mijenjati bez narušavanja mogućnosti klijenta za interakciju s web servisom. Čvrsto povezani sustav podrazumijeva da su logika klijenta i poslužitelja usko povezani jedni s drugom što implicira da ako se jedno sučelje promijeni, mora se ažurirati i drugo. Usvajanje lagano povezane arhitekture teži tome da softverski sustavi da se njima lakše upravlja i omoguće jednostavniju integraciju između različitih sustava
- Gruba zrnatost – izgradnja Java aplikacija zahtijeva stvaranje nekoliko sitnozrnih metoda koje se zatim sastavljaju u grubo zrnatu uslugu koju koristi klijent. Tehnologija web servisa pruža prirodan način definiranja grubo zrnatih usluga koje pristupaju poslovnoj logici.
- Sinkronost i asinkronost - Sinkronim pozivima web servis blokira klijenta sve dok servis ne završi sa započetom akcijom. S druge strane, asinkrone operacije omogućuju klijentu da poziva servis i u čekanju odgovora izvršava druge akcije. Asinkroni klijenti dobivaju svoj rezultat kasnije, dok sinkroni klijenti dobivaju svoj rezultat nakon završetka usluge.
- RPC (engl. *Remote Procedure Calls*) – web servisi omogućuju klijentima da pozivaju procedure i metode udaljenih objekata pomoću XML baziranom protokolu. Udaljene procedure izlažu ulazne i izlazne parametre koje web usluga mora podržavati.
- Razmjena dokumenata – web servisi podržavaju transparentnu razmjenu dokumenata kako bi se olakšala poslovna integracija.

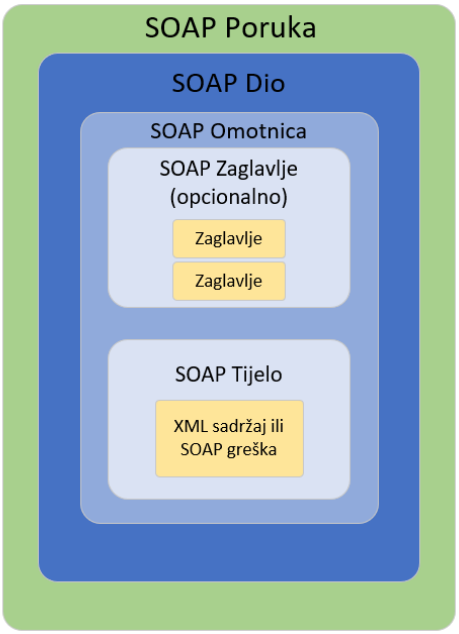
### 4.3.1. SOAP

SOAP (engl. *Simple Object Access Protocol*) pruža jednostavan i lagan mehanizam za razmjenu strukturiranih informacija između odvojenih komponenata u decentraliziranom, distribuiranom okruženju pomoću XML-a [26].
Postupak poziva metoda web servisa je izrazito važan, stoga mora biti dobro definiran. SOAP definira način razmjene poruka između pružatelja usluga i potrošača. Neke od glavnih osobina protokola su pouzdanost, sigurnost, povezanost, usmjeravanje i jednostavnost. U procesu slanja poruka postoje tri glavna sudionika: pošiljatelj, primatelj i posrednik. Pošiljatelj generira i šalje poruku, a primatelj prima istu poruke te generira odgovor ili grešku ako je došlo do nepredviđenih događaja. Posrednik je ujedno pošiljatelj i primatelj. Njegovo zaduženje je primanje i obrada SOAP zaglavlja te ponovno slanje SOAP poruke do izvornog pošiljatelja [29].

Prema službenoj dokumentaciji organizacije W3 [26]**,** SOAP se sastoji od tri djela:

- SOAP omotnica definira sadržaj poruke, primatelja poruke i obaveznost poruke.
- Pravila konstrukcije definiraju mehanizam serijalizacije poruke koji se koristi za razmjenu ranije definiranih tipova podataka.
- SOAP RPC definira konvencija koja se može koristiti za udaljenih proceduralnih zahtjeva i odgovora.

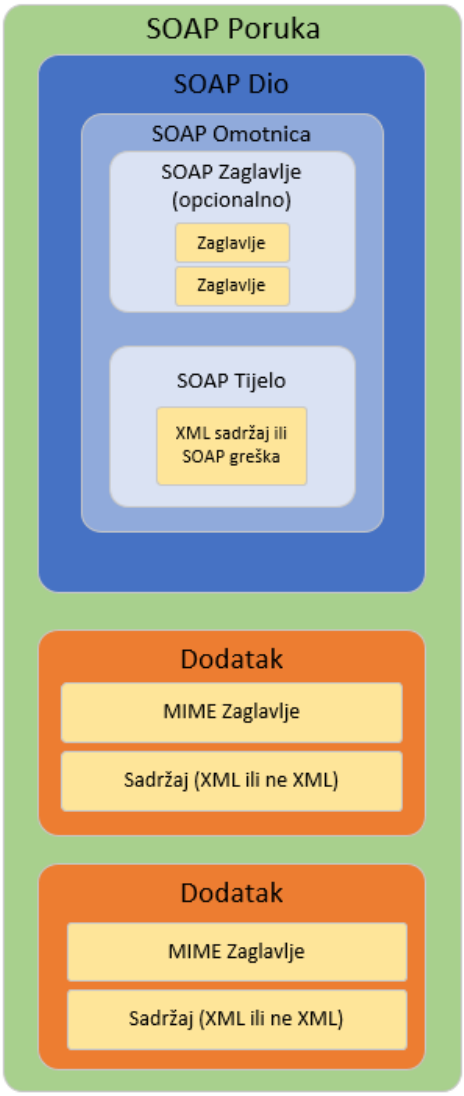
Kao što je navedeno, jedan od tri glavna dijela SOAP-a je omotnica. Omotnica je dio SOAP poruke koja je prikazana na slici 16.




Slika 16: SOAP poruka [27]

SOAP poruka predstavlja dobro strukturirani XML dokument koji sadrži ranije spomenutu omotnicu (engl. *Envelope*). Omotnica sadrži opcionalno zaglavlje i obavezno tijelo poruke. Tijelo sadrži jednostavan mehanizam za razmjenu obaveznih podataka namijenjenih krajnjem primatelju poruke te se samim time smatra glavnim dijelom SOAP poruke. Opcionalni dio omotnice je zaglavlje koje omogućuje jednostavno dodavanje novih protokola, autentifikacije i autorizacije, upravljanja transakcijama, obrada plaćanja, praćenje i revizija, i slično [28].

SOAP pruža i verziju poruke s dijelom za dodatak (engl. *AttachmentPart*) koji daje pruža mogućnost dodavanja informacija i podataka SOAP poruci. Kako se radi o opcionalnom dijelu SOAP poruke, korisnik mora kreirati dodatak i integrirati ga unutar poruke. Ako SOAP poruka sadrži dodatak, tada mora sadržavati i posebno zaglavlje. Radi se o MIME (engl. *Multipurpose* Internet *Mail Extensions*) zaglavlju koje indicira da se radi o poruci s dodatkom. Svaki dodatak unutar poruke mora sadržavati korespondirajuće MIME zaglavlje. Slika 17 predstavlja proširenu SOAP poruku [27].




Slika 17: Proširena SOAP poruka [27]

Jedan od usko povezanih i vrlo bitnih pojmova vezanih za SOAP jest WSDL. WSDL (engl. *Web Services Description Language*) je jezik za opis sučelja kojim se opisuje skup operacija koje podržava web servis, ulazne i izlazne objekte operacije, sheme podataka i načine povezivanja i konverzije podataka. Također, WSDL omogućuje automatsko generiranje koda klijenata web servisa temeljem svojeg sadržaja. Takav pristup automatskog generiranja web servisa iz WSDL popularno se naziva *Contract-First*. Konkurentski pristup, *Code-First podrazumijeva pisanje web servisa* te generiranje WSDL na temelju servisa [28].

JAX-WS API je specifikacija za programski jezik Java koja olakšava izradu web servisa i klijenata koji komuniciraju pomoću XML-a. Specifikacija omogućuje programerima da kreiraju web servise koji su orijentirani poruka i udaljenim pozivima (RPC). JAX-WS pozive metoda web servisa izvršava pomoću XML baziranog protokola, najčešće SOAP. Pravilno

strukturirane SOAP poruke prenose od pošiljatelja do servisa i obratno pomoću HTTP. Iako su same po sebi SOAP poruke kompleksne, JAX-WS API skriva složenost. Na poslužiteljskoj strani određuju se operacije web servisa definirajući metode sučelja napisanog u programskom jeziku Java. Klijentske programe je također lako implementirani. Klijent kreira proxy, lokalni objekt koji predstavlja uslugu, a zatim jednostavno poziva metode na proxy-ju. S JAX-WS, programer ne generira ili raščlanjuje SOAP poruke, već JAX-WS radni stroj pretvara API zahtjeve i odgovore u i iz SOAP poruka što bitno pojednostavljuje korištenje SOAP poruka [30].

### 4.3.2. REST

Pojam REST (engl. *REpresentational State Transfer*) uvodi Roy Fielding 2000. godine u svojoj doktorskoj disertaciji [31]. REST ne predstavlja arhitekturu već skup ograničenja koja primjenom na određeni softver stvaraju softver koji sadrži komponente, hiperlink-ove, uređene podatke, komunikacijske protokole i podatkovne potrošače [31].

Ograničenja koja sustav mora zadovoljiti kako bi se smatrao REST [31]:

- Mora imati klijent-server arhitekturu
- Zahtjevi za podacima međusobno moraju biti neovisni
- Mora pružati podršku za cache na više razina
- Svaki resurs mora imati unikatnu adresu i točku pristupa
- Mora biti višeslojan
- Opcionalna mogućnost dohvata programskog koda na zahtjev

Kada se spominje REST najčešće ga se povezuje s HTTP. Iako REST nije vezan ni za jedan protokol u većini slučajeva koristi se HTTP. HTTP je jednostavan sinkroni internet protokol baziran na zahtjev/odgovor principu. Protokol se primjenjuje za distribuirane, kolaborativne sustave temeljene na dokumentima. Velika prednost HTTP-a je u tome što tijelo poruke može sadržavati bilo koji format poruke. Neki od najpopularnijih su HTML, XML, JSON, običan tekst i binaran tip podatka [32].

Web aplikacije koje slijedi REST zahtjeve nazivaju se RESTful web servisi. Takvi servisi svoje funkcionalnosti temelje na HTTP metodama i moraju biti u skladu s definicijom HTTP-a. Metode koje HTTP definira su:

- POST – koristi se kod kreiranja resursa
- GET– koristi se za dohvat resursa
- PUT – koristi se za promjenu stanja resursa

- DELETE – koristi se za uklanjanje resursa

U današnje vrijeme većina novih projekata svoje servise izrađuju kao REST servisi. Neki od najpoznatijih tvrtki kao što su Facebook, Google i Twitter svoje poslovanje temelje ne RESTful web servisima [29].

JAX-RS slično kao JAX-WS, predstavlja specifikaciju za lakše kreiranje web servisa, samo ovoga puta RESTful servisa. JAX-RS API koristi anotacije, koje su sastavni dio Java od 7 verzije, za lakšu izradu RESTful web servisa. Razvojni programeri prilikom izrade novog web servisa pomoću anotacija definiraju resurse i akcije koje se mogu izvršiti nad definiranim resursima. Aplikacija koja koristi JAX-RS imati će automatski konfigurirane resurse, pomoćne klase, te resurs koji je potrebno isporučiti na poslužitelj kako bi klijenti mogli koristiti web servis [33].

Tablica 3 sadrži neke od najpopularnijih anotacija za izradu i definiranje RESTful web servisa.

Tablica 3: JAX-RS anotacije [33]

| Naziv | Opis |
|---|---|
| @Path | Anotacija @Path predstavlja relativnu URI putanja koja određuje mjesto na kojem će biti smještene Java klase, npr `/helloworld`. Vrlo korisna mogućnost anotacije @Path je dodavanje varijabli u URI putanju, npr. `helloworld/{korisnik}` |
| @GET | @GET anotacija je predviđena za označavanje metoda, a korespondira HTTP metoda jednakog naziva. Java metoda označena ovom anotacije obrađuje HTTP GET zahtjeve. |
| @POST | @POST anotacija je predviđena za označavanje metoda, a korespondira HTTP metoda jednakog naziva. Java metoda označena ovom anotacije obrađuje HTTP |

| | |
|---|---|
| | POST zahtjeve. |
| @PUT | @PUT anotacija je predviđena za označavanje metoda, a korespondira HTTP metoda jednakog naziva. Java metoda označena ovom anotacije obrađuje HTTP PUT zahtjeve. |
| @DELETE | @DELETE anotacija je predviđena za označavanje metoda, a korespondira HTTP metoda jednakog naziva. Java metoda označena ovom anotacije obrađuje HTTP DELETE zahtjeve. |
| @PathParam | Anotacija @PathParam je tip anotacije koji se koristi za označavanje resursa unutar klase. Parametri URI putanje izvlače iz URI zahtjeva, a nazivi parametara korespondiraju nazivu URI varijable specificirane @Path anotacijom |
| @QueryParam | Anotacija @QueryParam koristi se za anotiranje resursa. Vrijednost @QueryParam dobiva se iz URI parametara upita |
| @Consumes | Anotacija @Consumes koristi se za definiranje MIME vrsta medija koje resurs može konzumirati, a proslijeđeni je od strane klijenta, npr. `application/json` |
| @Produces | Anotacija @Produces koristi se za definiranje MIME vrsta medija koje resurs može proizvesti i poslati klijentu, npr. `text/plain` |
| @ApplicationPath | Anotacija @ApplicationPath koristi se za definiranje URL mapiranja aplikacije. Putanja definirana putem ove anotacije je osnovni URI za sve URI specificirane pomoću @Path anotacije. |

### 4.3.3. Usporedba web servisa

Karakteristike SOAP i REST servisa međusobno se razlikuju. Tablica 4 prikazuje usporedbu karakteristika servisa.

Tablica 4: Usporedba karakteristika SOAP i REST web servisa [34]

| SOAP | REST |
|---|---|
| Dobro poznata, tradicionalna tehnologija | Nova tehnologija u usporedbi sa SOAP |
| Unutar B2B poslovanja SOAP je i dalje popularan | Postoje razna područja unutar koji se REST primjenjuje u produkcijske svrhe. Jedno on najbitnijih je bankarski sektor |
| U SOAP-u je interakcija klijent-poslužitelj čvrsto povezana. | U REST-u interakcija klijent-poslužitelj je slabo povezana. |
| Izmjena u SOAP servisu često podrazumijeva ponovno kompiliranje programskog koda te izmjene na klijentskoj strani | Izmjene na REST servisu ne zahtijevaju nikakve izmjene na klijentskoj strani |
| Podržava slanje dokumenata isključivo u binarnom formatu | Podržava direktno slanje svih tipova podataka |
| SOAP nema stabilno podršku za bežičnu (engl. *Wireless*) infrastrukturu | Ima dobro podršku za bežičnu infrastrukturu |
| SOAP servis uvijek kao rezultat poziva metoda vraća podatke tipa XML | REST servisi podržavaju različite tipove podataka u odgovoru metode |
| Odgovor SOAP servisi po veličini zahtjeva do 10 puta više byte-ova od REST servisa što rezultira sporijim odgovorom | Odgovori su brži – poruke zauzimaju manje memorije u odnosu na SOAP servise |
| SOAP zahtjevi koriste POST metode koji zahtijevaju kreiranje kompleksnih XML zahtjeva koje je teško spremiti u cache memoriju | REST zahtjevi mogu ići preko GET metode koje je jednostavno spremiti u cache memoriju |
| Dizajnirati za korištenje unutar distribuiranog okruženja | Pretpostavlja komunikaciju od točke do točke (engl. *point-to-point*), nije predviđen za udaljena okruženja |
| Težak za izradu, pretpostavlja korištenje | Puno jednostavnija izrada u odnosu na |

| | |
|---|---|
| dodatnih alata | SOAP servise |
| Kriva predodžba da je SOAP sigurniji | REST pretpostavlja prijenos preko HTTP ili HTTPS i dodatno raspoloživi ugrađeni protokoli |
| Trenutno se više koristi i nudi bolju integraciju s drugim alatima i podršku dobavljača | Nedostatak podrške za sigurnosne standarde i poruke |

Nakon predstavljene usporedbe karakteristika SOAP i REST servisa, u nastavku su prikazani grafovi usporedbe performansi SOAP i REST servisa dobivenih unutar istraživanja „*Comparing Performance of Web Service Interaction Styles: SOAP vs. REST*“ [35]. Grafovi prikazuje vrijeme odaziva po broj zahtjevu u dva scenarija: interneta veza preko žica i bez žice.




Slika 18: SOAP vs REST - Žičana veza [35]




Slika 19: SOAP vs REST - Bežična veza [35]

Iz prikazanih grafova vidljivo je da u žičanom scenariju SOAP web servis ostvaruje znatno gore rezultate u odnosu na REST. Također, vidljivo da se s povećanjem zahtjeva vrijeme odaziva bitno povećava kod SOAP servisa. U bežičnom scenariju oba servisa ima gore rezultate u odnosu na žičani scenarij, ali REST servisi i dalje imaju nešto brže vrijeme odaziva na većem broj zahtjeva.

Temeljem dobivenih rezultata, autori zaključuju da su REST servisi skalabilniji, interoperabilniji i boljih performansi u odnosu na SOAP servise, dok s druge strane SOAP servisi pružaju veći stupanj sigurnosti i pouzdanosti [35].

# 5. Implementacija sustava

Nakon teorijskog opisa IoT-a, MQTT protokola te drugih metoda komunikacije između udaljenih sustava, unutar ovog poglavlja pomoću praktičnog projekta objedinjene su ranije opisane metode i tehnologije u potpuno funkcionalan sustav. U nastavku rada slijedi detaljan opis implementiranog sustava i tehnologija koje su korištene, prikaz arhitekture sustave, te izgled same aplikacije i najbitniji dijelovi programskog koda.

## 5.1. Opis i funkcionalnosti sustava

Implementirani sustav naziva Sustav za upravljanje IoT uređajima, kao što mu i ime govori, primarno služi za upravljanje povezanim uređajima. Sustav se sastoji od dvije aplikacije s grafičkim sučeljem naziva: Korisnički portal i Admin aplikacija. Srž sustava čine IoT uređaji koji rade kao međusobno odvojene komponente. Svaki od dostupnih uređaja u pravilnom intervalu očitava podatke sa senzora i šalje ih prema MQTT posredniku. Osim očitanih podataka, IoT uređaj šalje interval očitavanja i status prema MQTT posredniku. Unutar aplikacije postoji EJB (engl. *Enterprise JavaBean*) modul koji je implementiran kao zasebna komponenta koja je zadužena za rad s MQTT porukama. Svaka poruka na koju se bilo koja komponenta sustava prijavila dolazi do EJB modula koji ju dalje prosljeđuje. Također, prijava na određenu temu poruka također se izvršava pomoću EJB modula.

MQTT posrednik je odvojena komponenta koji služi za upravljanje MQTT porukama. Unutar sustava MQTT posrednik je implementiran na IoT uređaju naziva Raspberry PI te se radi o Mosquitto implementaciji posrednika koja je ranije opisana.

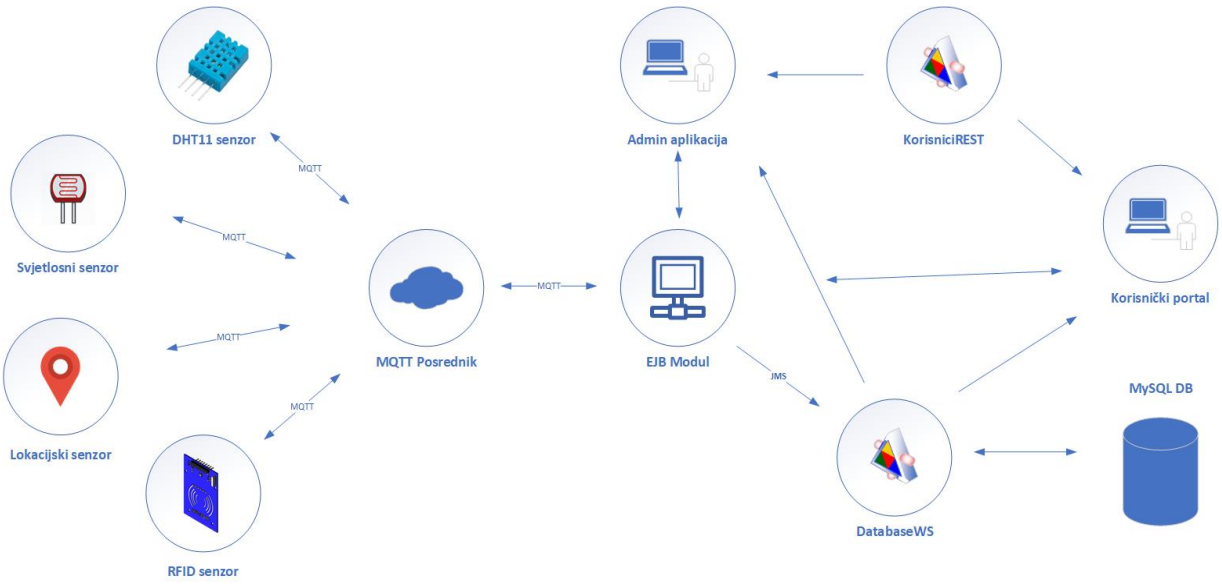
Admin aplikacija omogućuje upravljanje IoT uređajima. Svakom IoT uređaju administrator sustava može promijeniti interval objave kao i pauzirati uređaj odnosno aktivirati ovisno o prethodnom stanju. Također, postoji mogućnost pauziranja svih uređaja unutar sustava i promijene intervala svim uređajima. Osim mogućnosti upravljanja IoT uređajima, administrator ima uvid u sve aktivne uređaje kao i sve MQTT poruke koje su primljene unutar cijelog sustava. Što se tiče korisnika i korisničkih akcija, administrator ima uvid u sve registrirane korisnike kao i dnevnik aktivnosti korisnika unutar cijelog sustava. Posljednja mogućnost koju administrator ima na raspolaganju jest MQTT konzola. MQTT konzola omogućuje administratoru upravljanje pojedinim uređajem ručnim slanjem poruka kao i testiranje cijelog sustava kroz grafičke komponente konzole.

Druga aplikacija s korisničkim sučeljem, naziva Korisnički portal, daje korisnicima uvid u trenutno očitane podatke s IoT senzora. Svaki dostupan uređaj ima svoju odvojenu web stranicu gdje korisnik može pratiti trenutne podatke, ali i povijesne podatke za pojedini uređaj. Korisnik ima i mogućnost ažuriranja profila ako je potrebno.

Sustav za upravljanje IoT uređajima sastoji se od dva web servisa. Prvi servis, naziva KorisniciREST servis, zadužen je za sve akcije vezane za korisnike. Pomoću navedenog servisa registriraju se novi korisnici, autentificiraju postojeći prilikom prijave u sustav, korisnici ažuriraju svoje profile te administratori nadgledaju registrirane korisnike sustava.

Drugi servis, naziva DatabaseWS, je SOAP servis koji je zadužen za CRUD operacije na podacima koji su očitani sa senzora, primijenim MQTT porukama te dostupnim IoT uređajima. DatabaseWS ostvaruje vezu prema MySQL bazi podatka koji sadrži sve arhivirane podatka sustava.

Slika 20 predstavlja opisane komponente te međusobnu komunikaciju između komponenta Sustava za upravljanje IoT uređajima.




Slika 20: Komponente Sustava za upravljanje IoT uređajima (autorski rad)

## 5.2. Korišteni alati i tehnologije

Sustav za upravljanje IoT uređajima razvijen je pomoću dva razvojna okruženja. Prvi dio, vezan za web servise i aplikacije sustava, razvijen je pomoću Netbeans razvojnog okruženja koristeći programski jezik Java, točnije Java EE. Kako se radi o sustavu koji svoju logiku temelji na web servisima i web aplikacijama potreban je poslužitelj za isporuku aplikacija. Kao poslužitelj odabran je Glassfish 5.5. Radi se o poslužitelju otvorenoga koda (engl. *Open Source*), koji je napisan u programskom jeziku Java, a namijenjen isporuci gotovih aplikacija. Baza podataka koju koristi sustav je MySQL. Neke od poznatijih Java biblioteka i razvojnih okvira koji su također korišteni unutar projekta:

- Eclipse Jersey - REST razvojni okvir koji pruža JAX-RS implementaciju
- JAX-WS - razvojni okvir koji pruža alate i infrastrukturu za implementaciju web servisa
- JSF (engl. *JavaServer Faces*) - Java razvojni okvir namijenjen za izradu korisničkog sučelja
- PrimeFaces - besplatan razvojni okvir za JSF koji sadrži brojne gotove komponente prikladne za izradu korisničkog sučelja
- Project Lombok – besplatna Java biblioteka koja omogućava davanje osobina programskim varijablama pomoću anotacija
- MqttClient – besplatna Java biblioteka koja pruža za MQTT

Drugi dio sustava vezan za IoT uređaje razvijen je pomoću Arduino IDE razvojnog okruženja. Arduino IDE je također razvojni okvir otvorenog koda koji pojednostavljuje proces kompiliranja i prijenosa koda na IoT uređaje. Neke od poznatijih biblioteka koje su korištene za razvoj i konfiguriranje IoT uređaja:

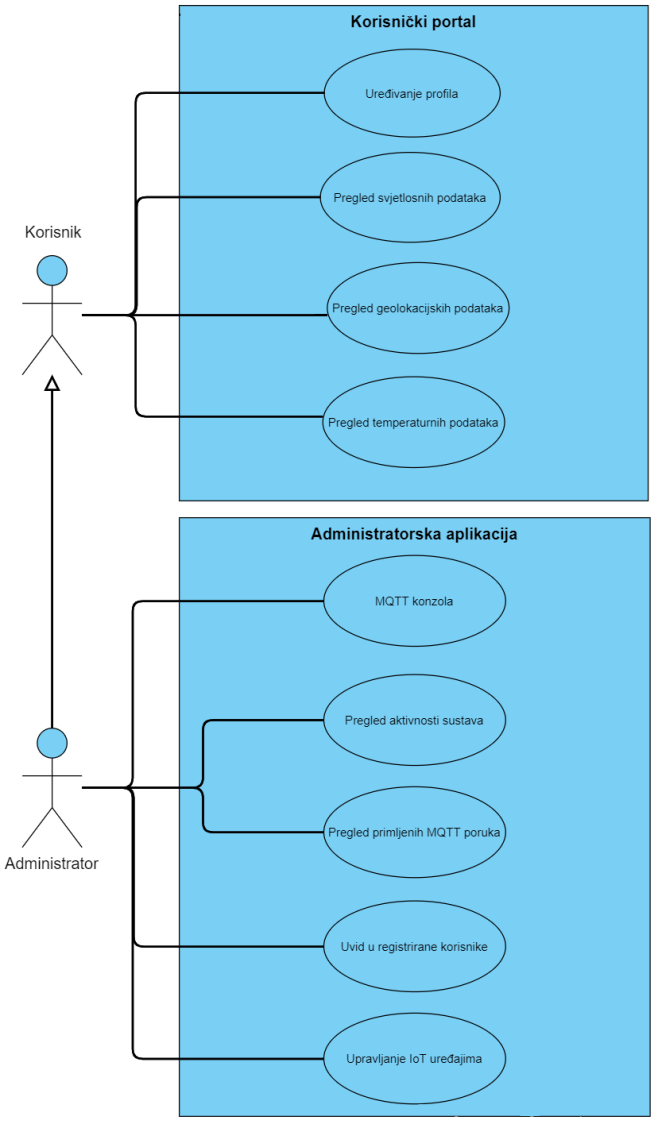
- WiFi – besplatna biblioteka koja omogućuje spajanje IoT uređaja na internet
- PubSubClient – besplatna biblioteka koja omogućuje slanje i primanje MQTT poruka
- DHT – besplatna biblioteka za seriju DHT uređaja
- WifiLocation – besplatna biblioteka koja implementira pozive prema Google Maps GeoLocation API s ciljem dobivanja lokacije iz WiFi mreže bez potrebe za korištenjem GPS prijemnika
- MFRC522 – besplatna biblioteka za MFRC522 RFID čitaće i pisače
- SPI – besplatna biblioteka za serijsku komunikaciju između IoT uređaja

## 5.3. Arhitektura sustava

### 5.3.1. Dijagram slučaja korištenja sustava

Opis funkcionalnosti koje sustav za upravljanjem IoT uređajima pruža prikazan je korištenjem dijagrama slučajeva korištenja. Na dijagramu nije prikazana prijava koju korisnik, ali i administrator moraju uspješno obaviti prije korištenja sustava te se podrazumijeva da je taj korak napravljen. Korisnik nakon prijave ima nekoliko mogućnosti. Prva od njih jest uređivanje profila, gdje korisnik ažurira vlastite podatke. Sljedeća mogućnost koja je korisniku dostupna jest pregled podataka sa senzora. Navedena mogućnost podijeljena je u tri podskupine za svaki dostupan senzor. Korisniku na mogućnosti stoje pregled podataka sa senzora za temperaturu, svjetlost i geolokaciju.

Administrator nakon uspješne prijave na raspolaganju ima pet različitih mogućnosti. Tri mogućnosti odnose se na povijesni uvid aktivnosti unutar sustava: pregled aktivnosti sustava, uvid u registrirane korisnike i pregled primljenih MQTT poruka. Najvažnija mogućnost administratora jest upravljanje IoT uređajima, gdje administrator prema volji može kontrolirati svaki dostupan IoT uređaj. Posljednja mogućnost koja je dostupna administratoru jest MQTT konzola koja omogućava upravljanje MQTT porukama prema potrebi.



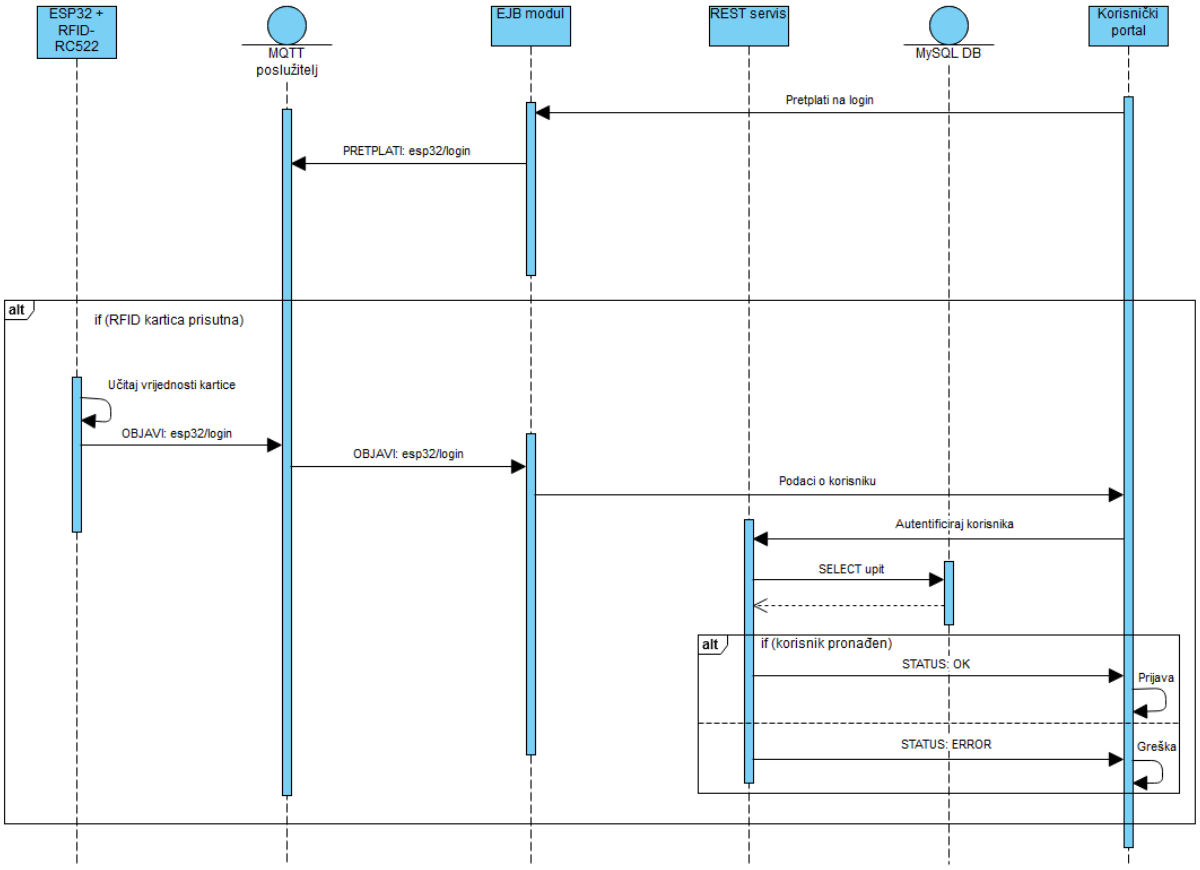

Slika 21: Dijagram slučaja korištenja sustava

### 5.3.2. Dijagram slijeda aktivnosti

Dijagramom slijeda aktivnosti prikazane su tri najbitnije aktivnosti sustava: prijava u sustav pomoću IoT uređaja, pregled podataka očitanih s IoT uređaja i upravljanje samim IoT uređajem.

Prvi dijagram slijeda aktivnosti predstavlja aktivnosti za prijavu u sustav pomoću IoT uređaja. Aktivnosti počinju pretplatom Korisničkog portala na MQTT poruke za prijavu u sustav. Kao i sve ostale aktivnosti vezane za MQTT poruke, pretplata se odvija pomoću EJB modula. Nakon što je EJB modul napravio pretplatu kod MQTT poslužitelja, sustav je spreman za prijavu pomoću MQTT sustava. Drugi dio aktivnosti počinje s IoT uređajem. ESP32 uređaj

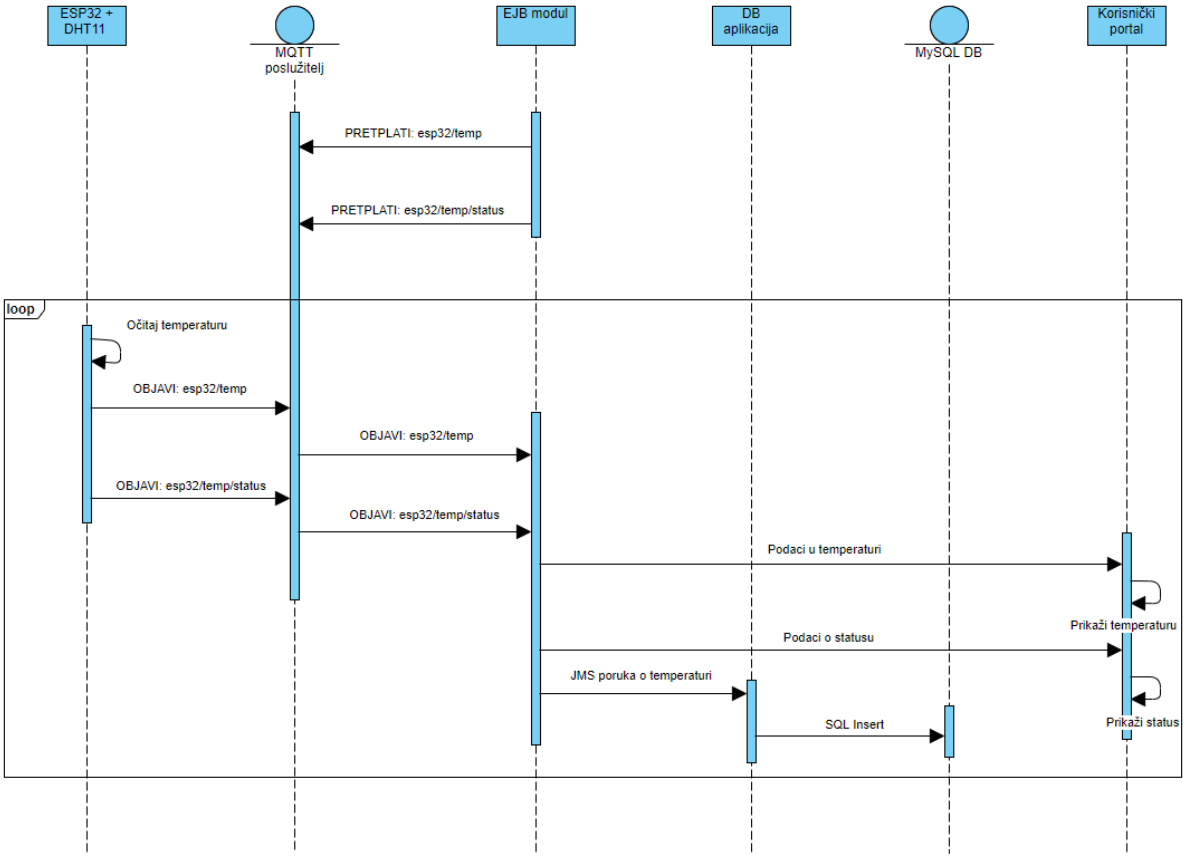
koji je povezan s RFID senzorom u intervalu od jedne sekunde provjerava je li prisutna RFID kartica s podacima za prijavu. U slučaju da uređaj pronađe valjanu karticu, šalje MQTT poslužitelju poruku koja sadrži potrebne podatke za prijavu. Nakon što MQTT poslužitelj primi poruke, prosljeđuje ju ranije pretplaćenom EJB modulu koji potom obavještava Korisnički portal. Nakon što Korisnički portal primi obavijesti o pokušaju prijave pomoću IoT uređaja, šalje obavijest REST servisu koji se zadužen za autentifikaciju korisnika. REST servis provjerava postoji li korisnik s proslijeđenim podaci unutar baze podataka. U slučaju da postoji, REST servis vraća pozitivan odgovor Korisničkom portalu koji potom prijavljuje korisnika u sustav. U slučaju da korisnik nije pronađen, Korisnički portal ispisuje grešku o neuspješnoj prijavi pomoću IoT uređaja.




Slika 22: Dijagram slijeda aktivnosti - Prijava u sustav

Drugi dijagram slijeda predstavlja aktivnosti za prikaz podataka očitanih s IoT uređaja. Ovaj dijagram konkretno prikazuje ESP32 uređaj povezan s DHT11 uređajem za očitavanje temperature i vlage. Uređaji za očitavanje svjetlosti i geolokaciju imaju vrlo sličan slijed aktivnosti. Prilikom pokretanja sustava EJB modul se automatski pretplaćuje kod MQTT

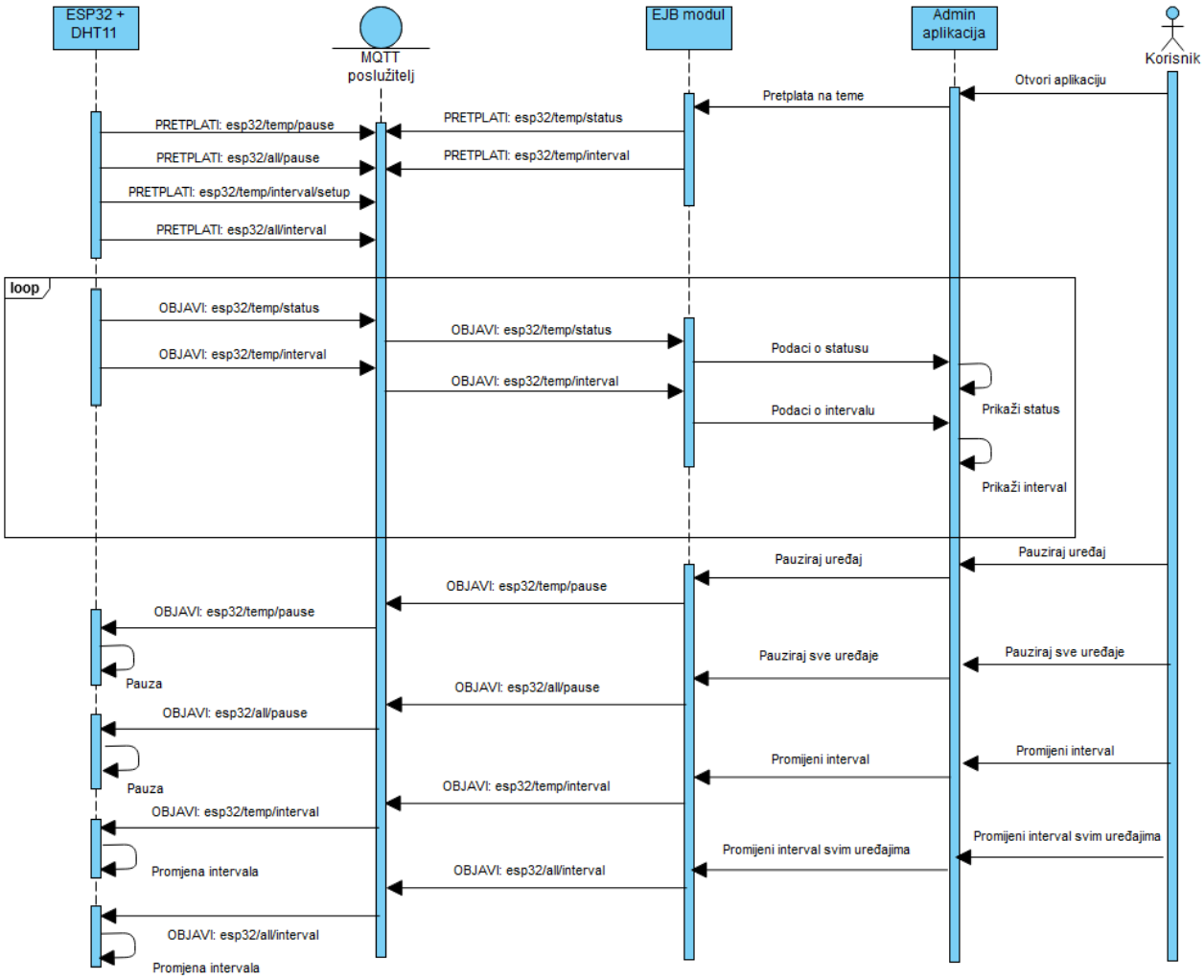
poslužitelja na poruke vezane za očitanu temperaturu i status IoT uređaja. IoT uređaj u pravilnom intervalu zadanom pomoću konfiguracije očitava temperaturu sa senzora te ju šalje kao MQTT poruku prema MQTT poslužitelju. Kao drugu poruku šalje status uređaja. Nakon što MQTT poslužitelj primi poruke prosljeđuje ih ranije pretplaćenom EJB modulu koji ih potom prosljeđuje Korisničkom portalu. Nakon što Korisnički portal primi obavijest o novoj temperaturi i status, dobivene podatke prikazuje korisniku. EJB modul osim slanja obavijesti Korisničkom portalu, šalje JMS poruku sa sadržajem MQTT poruke aplikaciji koja je zadužena za vezu prema bazi podataka koja potom poruku sprema u bazu podatka kako bi kasnije bila dostupna za pregled.




Slika 23: Dijagram slijeda aktivnosti - Prikaz podataka očitanih s IoT uređaja

Zadnji dijagram slijeda aktivnosti predstavlja složenu aktivnost upravljanja IoT uređajem. Kao i kod prošlog dijagrama, na slici broj 23 prikazan je slijed aktivnosti za ESP32 uređaj i DHT11 senzor koji mjeri temperaturu, a slijed aktivnosti za ostale IoT uređaje koji mjere drugu vrstu podataka je vrlo sličan. Aktivnosti započinju korisničkom prijavom u Admin aplikaciju nakon čega Admin aplikacija obavještava EJB modul o potrebi za pretplatom na teme vezane za temperaturu. EJB modul pretplaćuje se kod MQTT poslužitelja na teme vezane za status i

interval objavljivanja uređaja za mjerenje temperature. S druge strane, IoT uređaje se pretplaćuje kod MQTT poslužitelja na teme za pauzu i promjenu intervala specifičnog uređaja, ali i tema za sve dostupne uređaje unutar sustava. U intervalu od jedne sekunde IoT uređaje šalje MQTT poruke poslužitelju s podacima o statusu uređaja i intervalu objave očitanih podataka. Nakon što MQTT poslužitelj primi poruke, prosljeđuje ih ranije pretplaćenom EJB modulu koji obavještava Admin aplikaciju o pristiglim poruka. Kada Admin aplikacija primi obavijest o pristiglim podacima, iste podatke prikazuje korisniku kroz korisničko sučelje. Korisnik kroz sučelje na raspolaganju ima četiri različite upravljačke naredbe za svaki uređaj. Prva moguća naredba je pauziranje uređaja kojim se onemogućuje daljnje slanje očitanih vrijednosti. Druga naredba je promjena intervala objave očitanih vrijednosti. Zadnje dvije naredbe odnose se također na pauzu i promjenu intervala uređaja, ali u ovom slučaju utječu na sve IoT uređaje koji su povezani unutar sustava.

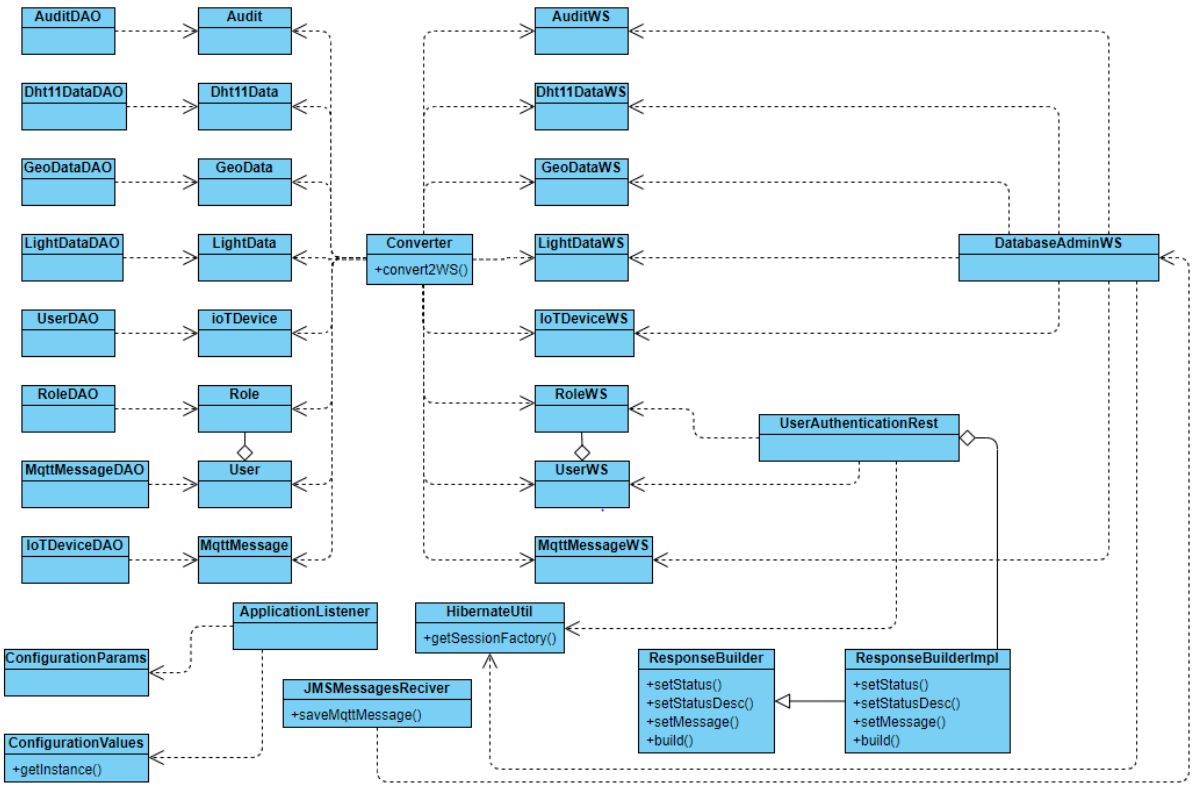



Slika 24: Dijagram slijeda aktivnosti - Upravljanje IoT uređajem

### 5.3.3. Dijagram klasa

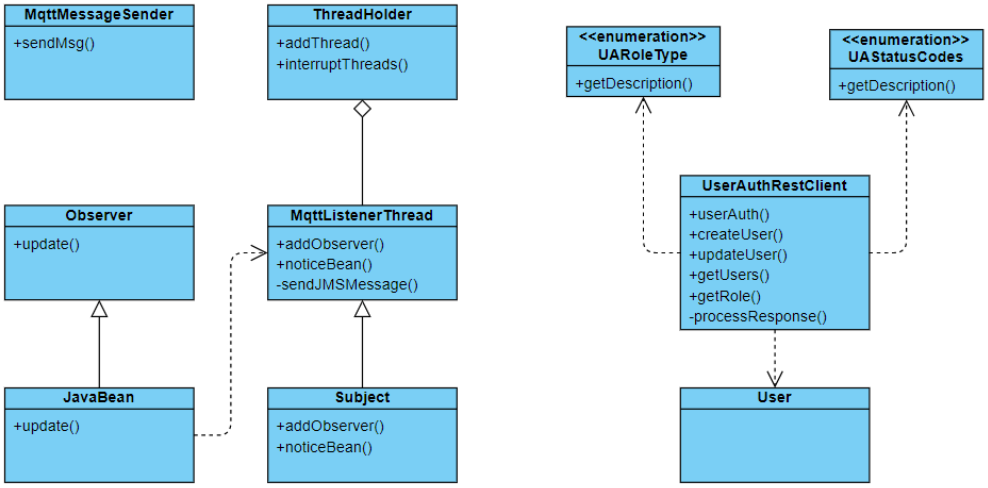
Kao što je ranije opisano, arhitektura sustava za upravljanjem IoT uređaja sastoji se četiri odvojene aplikacije. Dvije od četiri dostupne aplikacije zadužene su za prikaz podataka korisniku te ne sadrže posebnu programsku logiku. Sukladno navedenom, izrađena su dva dijagrama klasa za aplikaciju za vezu prema bazi podataka kao i EJB modul.

Prvi dijagram klasa predstavlja klase potrebne za rad aplikacije za vezu prema bazi podataka. Većina dostupnih klasa odnosi se na podatkovne klase koje prema strukturi odgovaraju tablicama unutar baze podataka. Navedene podatkovne klase se pomoću objektno-relacijskog alata za mapiranje Hibernate preslikavaju u tablice baze podataka. Za svaku podatkovnu klasu postoji i njena preslika koja ima sufiks WS. Takve klase namijenjene su za rad s web servisima. Na dijagramu klasa vidljive su i klase koje predstavljaju SOAP i REST servise te razne pomoćne klase za lakši rad aplikacije.




Slika 25: Dijagram klasa - DB aplikacija

Drugi dijagram klasa predstavlja klase sadržane unutar EJB modula. Dvije glavne klase ovog modula su `MqttMessageSender` i `MqttListenerThread`, koje služe za slanje odnosno primanje MQTT poruka. Kako se radi o relativno složenoj logici za MQTT poruke postoje pomoćne klase zbog lakšeg upravljanja porukama. Osim klasa za rad s MQTT porukama EJB modul sadrži i klasu koja predstavlja klijenta prema REST servisu aplikacije za vezu prema bazi podataka.

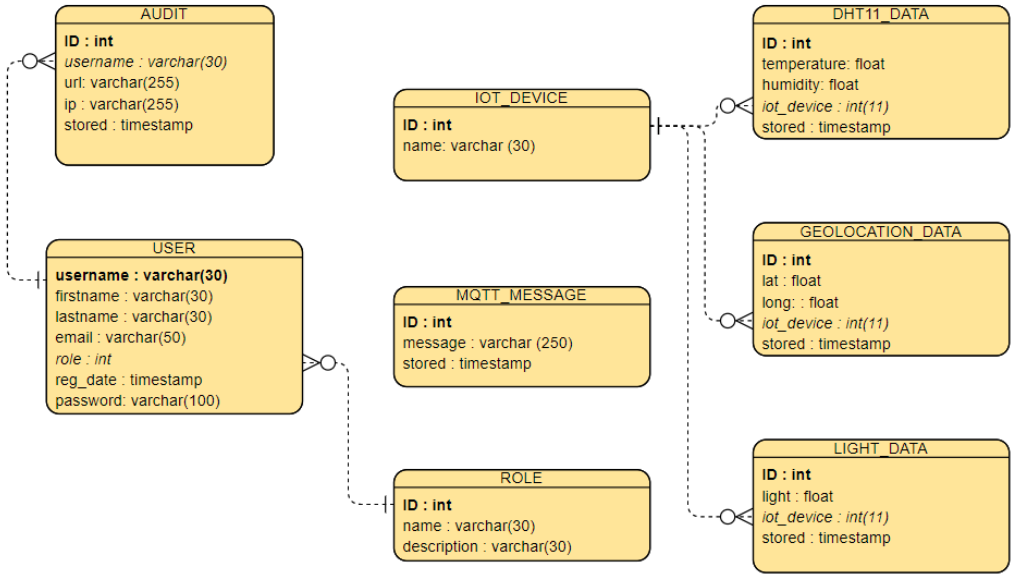



Slika 26: Dijagram klasa - EJB modul

### 5.3.4. ERA model

Na slici 27 nalazi se ERA model relacijske baze podatka korištene unutar Sustava za upravljanjem IoT uređajima. Model se sastoji od 6 odvojenih tablica. Tablica *User* predstavlja registrirane korisnike. Svaki korisnik unutar baze podatka mora imati dodijeljenu ulogu koja je pohranjena u tablici *Role*. Za praćenje korisničkih aktivnosti koristi se tablica *Audit* unutar koje je pohranjena akcija koju je određeni korisnik napravio unutar sustava.

Drugi dio modela predstavljaju podaci izmjereni pomoću IoT uređaja. Svaki dostupan uređaj unutar sustava pohranjen je u tablici naziva *IoT_device*. Također, podaci izmjereni preko senzora pohranjuju se u zasebnu tablicu. Tablica *MQTT_message* sadrži sve MQTT poruke koje su primljene unutar sustava.




Slika 27: ERA model sustava

## 5.4. Komponente sustava

### 5.4.1. IoT uređaji

Sustav za praćenje IoT uređaja sastoji se od više odvojenih uređaja. Svaki IoT predstavlja zasebnu komponentu sustava koja prikazuje mjerljive podatke iz svojeg fizičkog okruženja. Dodavanje novih kao i uklanjanje postojećih uređaja vrlo je jednostavno. Cijeli sustav građen je na principu uske povezanosti koja se temelji na niskoj povezanosti sustava s IoT uređajima.

Sustav se sastoji od četiri odvojena IoT uređaja kojima je glavna namjena mjerenje podataka i ranije opisanog Raspberry Pi uređaja koji ima ulogu MQTT posrednika. Svaki od četiri uređaja sastoji se od ESP32 mikro kontrolera i odgovarajućeg senzora za mjerenje podataka.

ESP32 mikro kontroler predstavljen je 2016. godine od strane Espressif System kao nasljednik popularnog ESP8266 mikro kontrolera. Glavne odlike ESP32 uređaja su niska cijena i potrošnja te integrirana podrška za Wi-Fi i Bluetooth. Također, ima mogućnost djelovanja kao kompletan samostalan sustav ili kao rob uređaj (engl. *Slave*) kompleksnog sustava [36].

Programski kod EPS32 uređaja pisan je pomoću Arduino IDE razvojnog okruženja te je učitan (engl. *Upload*) na EPS32 uređaj. Programski kod razvijen pomoću Arduino IDE razvojnog okruženja sastoji se od dvije odvojene funkcije: postavi (engl. *Setup*) i izvodi (engl. *Loop*). Prvi metoda, naziva postavi, izvršava se jednom za svaki uređaj i to prilikom inicijalizacije uređaja. Svrha metode je postavljanje svih početnih parametara i instanciranje klasa i struktura koje su potrebne za daljnji rad uređaja. Slika 28 prikazuje primjer metode postavi. Unutar metode otvara se kanal za serijsku komunikaciju uređaja, postavlja se veza na internet, instanciraju se klase za očitavanje podataka sa senzora i MQTT protokol te se uređaj prijavljuje na MQTT teme.

```
void setup() {
  Serial.begin(9600);

  setup_wifi();

  client.setServer(mqtt_server, 1883);

  dht.begin();

  Serial.println("Subscribed to: ");

  client.subscribe(topic_all_interval);
  Serial.println(topic_all_interval);
  client.subscribe(topic_all_pause);
  Serial.println(topic_all_pause);
  client.subscribe(topic_temp_pause);
  Serial.println(topic_temp_pause);
  client.subscribe(topic_temp_interval_setup);
  Serial.println(topic_temp_interval_setup);

  client.setCallback(callback);
}
```

Slika 28: Postavi metoda

Druga metoda, naziva izvodi, izvršava se ciklično. U većini slučajeva nakon jednog ciklusa glavna dretva programa se pauzira na određen broj milisekundi kako ne bi došlo do preopterećenja uređaja. Unutar metode uređaj izvršava svoju glavnu funkcionalnost. Slika 29 prikazuje programski kod izvodi metoda. Za svaki uređaj definiran je pravilan interval očitavanja podataka sa senzora te slanje istih kao poruke prema MQTT posredniku. Osim podataka očitanih sa senzora, uređaj šalje svoj trenutni status i interval objavljivanja prema MQTT posrednika. Kako web administrator sustava ima mogućnost upravljanja statusom i intervalom uređaja vrlo je bitno da uređaj može obavijestiti sve potrebne aplikacije o svojem trenutnom stanju.

```
void loop() {

  if (!client.connected()) {
    reconnect();
  }

  client.loop();

  if (isRunning) {

    if (seconds % publishingInterval == 0) {

      float h = dht.readHumidity();
      float t = dht.readTemperature();

      if (isnan(h) || isnan(t)) {
        Serial.println(F("Failed to read from DHT sensor!"));
      } else {

        Serial.print(F("Humidity: "));
        Serial.print(h);
        Serial.print(F("%  Temperature: "));
        Serial.print(t);
        Serial.print(F("°C "));

        String msg = String(t);
        msg.concat(";");
        msg.concat(String(h));

        publishMsg(topic_temp, msg.c_str());
      }
    }
  }

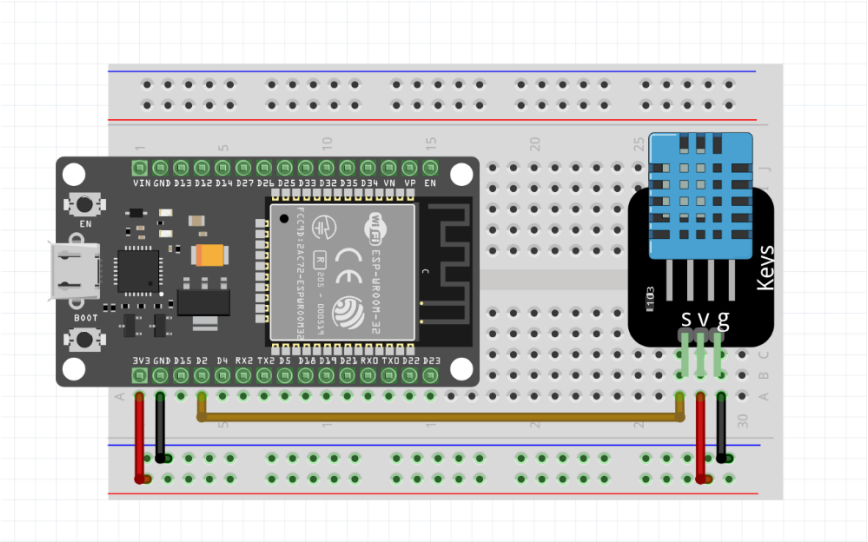
  publishMsg(topic_temp_status, String(isRunning).c_str());
  publishMsg(topic_temp_interval, String(publishingInterval).c_str());

  seconds += 1;
  delay(1000);

}
```

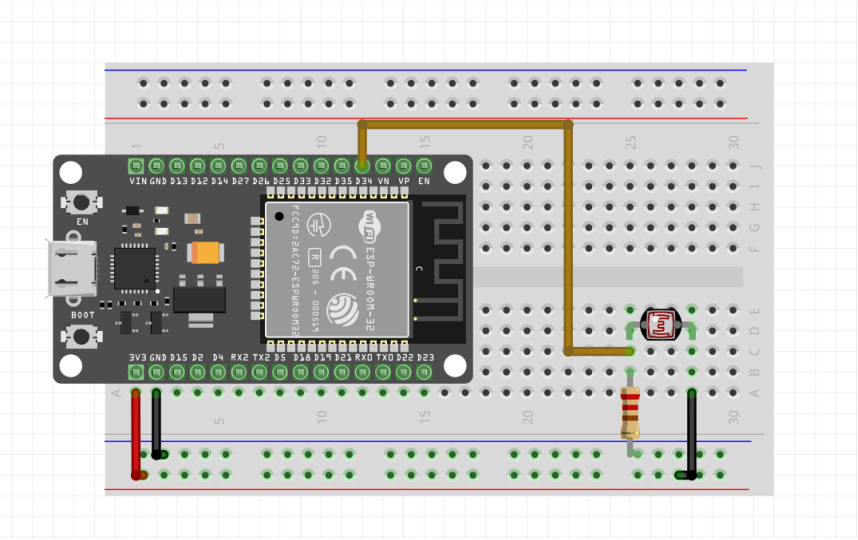
Slika 29: Izvodi metoda

Prvi IoT uređaj sastoji se od ESP32 mikro kontrolera i DHT11 senzora za mjerenje temperature i vlage. Uređaj u pravilnom intervalu očituje podatke o temperaturi i vlazi sa senzora te ih šalje prema MQTT posredniku. Također, na serijski izlaz ispisuje učitane podatke kako bi se mogao pratiti rad uređaja. Slika 30 predstavlja shemu spajanja navedenih komponenti.




Slika 30: Shema spajanja - ESP32 i DHT11 senzor

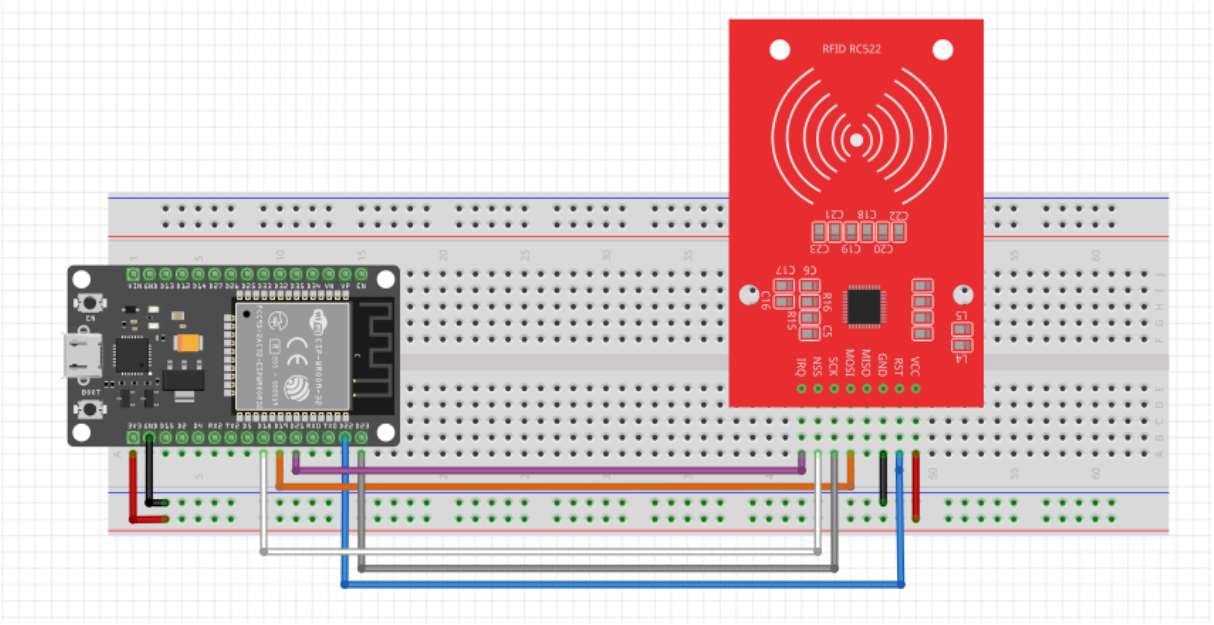
Drugi IoT uređaj sastoji se od ESP32 uređaja svjetlosnog otpornika koji ima mogućnost mjerenja razine svjetlosti. Izmjereni podaci šalju se kao MQTT poruke prema posredniku te se kasnije obrađuju unutar nadzorne aplikacije. Slika 31 prikazuje shemu spajanja ESP 32 uređaja i svjetlosnog otpornika.



Slika 31: Shema spajanja - ESP32 i svjetlosni senzor

Ranije opisana mogućnost prijave u sustav pomoću RFID kartice omogućena je IoT uređajem koji se sastoji od ESP32 mikro kontrolera i RFID-RC522 senzora. RFID-RC522 senzor u slučaju prisustva RFID kartice šalje identifikacijsku oznaku kartice zajedno s podacima za prijavu u nadzornu aplikaciju kao MQTT poruku. U slučaju da se radi o

autoriziranoj kartici i točnoj kombinaciji korisničkih podataka prijava je uspješna. Slika 32 predstavlja shemu spajanja ESP32 mikro kontrolera i RFID-RC522 senzora.



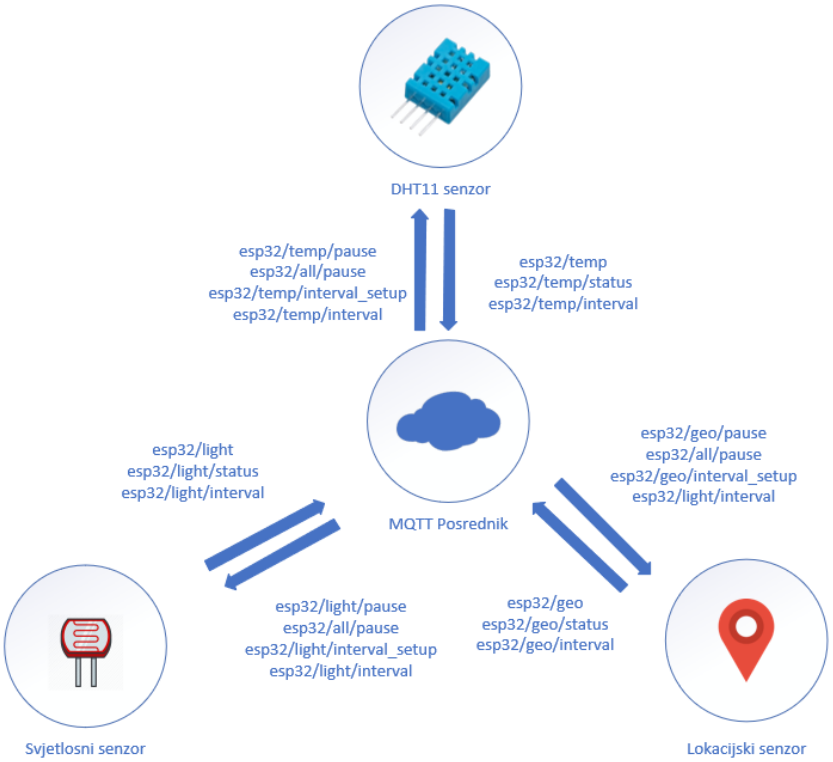

Slika 32: Shema spajanja - ESP32 i RFID-RC522 senzor

Zadnji IoT uređaj sastoji se isključivo od EPS32 mikro kontrolera. On u pravilnom intervalu pomoću GeoLocation API dohvaća trenutnu geografsku širinu i dužinu IoT uređaja te navedene podatke kao MQTT poruku šalje do posrednika koji dalje prosljeđuje poruke web aplikacijama.

## 5.4.2. MQTT poslužitelj

Jedna od glavnih komponenti Sustava za upravljanje IoT uređajima jest MQTT posrednik. Posrednik povezuje IoT uređaje s ostatkom aplikacije. Kao što je već ranije navedeno radi o Mosquitto posredniku instaliranom na Raspberry Pi Zero W uređaju. Raspberry Pi je jednostavan kompjuter u obliku računalne pločice razvijen primarno za školstvo, no primjena mu je česta kod IoT projekata. Instalacija MQTT posrednika na Raspberry Pi vrlo je jednostavna i kratkotrajna.

Kako se radi o sustavu s više IoT uređaja i više načina komunikacije postoje brojne teme na kojem se uređaji i aplikacije pretplaćuju odnosno šalju poruke. Slika 33 prikazuje način komunikacije IoT uređaja i MQTT posrednika.




Slika 33: Način komunikacije IoT uređaja i MQTT posrednika

Svaki uređaj objavljuje poruke u tri različite teme. Prva tema odnosi se na primarnu vrijednost koju IoT uređaj mjeri kao što je temperatura ili jačina svjetlosti, druga tema predstavlja status uređaja koji može biti aktivan ili pasivan, a treća se tema odnosi na interval objavljivanja. Svaki uređaj ima različite nazive spomenutih tema, kao što je vidljivo i iz slike, kako bi se točno znalo s kojeg je uređaja došao koji podatak.

S druge strane, svaki uređaja pretplati se na dvije teme koje su različite kod svakog uređaja. Prva tema odnosi na pauziranje uređaja, dok se druga odnosi na promjenu intervala.

Također, svi uređaji unutar sustava prijavljeni su na temu za pauziranje i promjenu intervala na razini sustava. Prijavom svih uređaja na iste teme ostvaruje se lakoća upravljanja više uređaja samo jednom porukom.

Drugi dio komunikacije odvija se između MQTT posrednika i EJB modula. Glavna zadaća EJB modula je primanje, odnosno slanje MQTT poruka. Nakon što modul primi poruku on obavještava web aplikaciju o pristigloj poruci nakon čega ju prosljeđuje. Ovakvom implementacijom smanjuje se povezanost cijelog sustava.

EJB modul se kod MQTT posrednika pretplaćuje na teme za vrijednost senzora, status i interval svakog pojedinog IoT uređaja unutar sustava. Nadalje, prijavljuje se na temu za prijavu pomoću IoT uređaja. Nakon što primi poruku putem jedne od ranije prijavljenih tema, poruku prosljeđuje u web aplikacije kojoj je ta poruka namijenjena, odnosno u neki slučajevima obje aplikacije.

Admin aplikacija preko EJB modula šalje poruke IoT uređajima. Za svaki od dostupnih uređaja Admin aplikacija može slati poruke za promjenu intervala i pauziranje uređaja. Isto tako, Admin aplikacija može slati ranije spomenute grupne poruke za pauziranje i promjenu intervala svih uređaja unutar Sustava za upravljanje IoT uređajima.

Slika 34 prikazuje tokove objave poruka i pretplate na poruke između web aplikacija, EJB modula i MQTT posrednika.




Slika 34: Način komunikacije EJB modula, MQTT posrednika i aplikacija

### 5.4.3. Baza podataka

Baza podataka jedna je od glavnih komponenata svakog složenog sustava. Korištenjem baze omogućuje se pohrana svih bitnih podataka za pravilan rad sustava, ali i povijesnih podataka koji su dostupni na korisnički zahtjev. Kao što je ranije navedeno Sustav za upravljanje IoT uređajima koristi relacijsku MySQL bazu podatka i objektno-relacijskog alata za mapiranje Hibernate. Hibernate omogućuje da se složene strukture relacijske baze podataka preslikaju u podatkovne Jave klase.

Definiranje Hibernate svojstava moguće je pomoću xml datoteka. Dvije glavne skupine datoteka su: konfiguracijske datoteke i datoteka mapiranja. Konfiguracijska datoteka obično se naziva *hibernate.cfg* i sadrži sve konfiguracijske postavke vezane za bazu podatka kao što su autentifikacijski podaci i driveri za povezivanje prema bazi. Svaka tablica baze podataka koja se želi mapirati na određenu Java klasu mora imati zasebnu datoteku. Ta datoteka opisuje načine mapiranja atributa klase i stupce tablice unutar baze podataka. Slika 35 prikazuje opisanu datoteku mapiranja. Prikazana datoteka povezuje tablicu *users* s Java klasom *Users*. Iz slike je vidljivo da je svaki atribut zasebno definiran kao i relacijske veze tablice.

Source History

```
1  <?xml version="1.0"?>
2  <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
3  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
4  <!-- Generated Aug 12, 2020 4:50:33 PM by Hibernate Tools 4.3.1 -->
5  <hibernate-mapping>
6      <class name="hr.foi.diplomski.db.objects.Users" table="users" catalog="devdb01" optimistic-lock="version">
7          <id name="username" type="string">
8              <column name="username" length="30" />
9              <generator class="assigned" />
10         </id>
11         <many-to-one name="role" class="hr.foi.diplomski.db.objects.Role" fetch="select">
12             <column name="role" not-null="true" />
13         </many-to-one>
14         <property name="firstname" type="string">
15             <column name="firstname" length="30" not-null="true" />
16         </property>
17         <property name="lastname" type="string">
18             <column name="lastname" length="30" not-null="true" />
19         </property>
20         <property name="email" type="string">
21             <column name="email" length="50" not-null="true" />
22         </property>
23         <property name="regDate" type="timestamp">
24             <column name="reg_date" length="19" not-null="true" />
25         </property>
26         <property name="password" type="string">
27             <column name="password" length="100" />
28         </property>
29     </class>
30 </hibernate-mapping>
```

Slika 35: Hibernate konfiguracijska datoteka

### 5.4.4.Web servisi

Web servisi čine glavni način prijenosa informacija unutar implementiranog sustava. Korištenjem servisa smanjuje se međusobna povezanost komponenti i dodaje se fleksibilnost cijelom sustavu. Također, servisi omogućuju povezivanje različitih tehnologija te izolaciju složene poslovne logike iza jednostavnih metoda web servisa.

Prvi web servis, naziva DatabaseAdminWS, SOAP je servis koji služi za interakciju s bazom podataka. Preko servisa klijenti mogu obavljati CRUD operacije nad tablicama baze podataka. Također, klijenti nemaju nikakve informacije o bazi podataka te pozadinskoj logici. Jedina komponenta koju klijenti koriste jest WSDL servisa koji im daje uvid u dostupne metode servise i potrebne strukture. Opisani web servis koriste web aplikacije za upravljanje podacima te EJB modul za spremanje pohranu pristiglih MQTT poruka.

SOAP web servis implementiran je korištenjem JAX-WS API specifikacije za programski jezik Java. JAX-WS API specifikacija implementira anotacije za definiranje samog servisa, metoda i ulaznih parametara servisa koje bitno olakšavaju implementaciju i korištenje web servisa. Prikaz metoda opisanog SOAP web servisa nalazi se na slici 36.

```
31    @WebService(serviceName = "DatabaseAdminWS")
32    public class DatabaseAdminWS {
33
34        @WebMethod(operationName = "createAudit")
35        public void createAudit(@WebParam(name = "auditWS") AuditWS auditWS) {...3 lines }
38
39        @WebMethod(operationName = "getAudits")
40        public List<AuditWS> getAllAudits() {...12 lines }
52
53        @WebMethod(operationName = "createRole")
54        public void createRole(@WebParam(name = "roleWS") RoleWS roleWS) {...3 lines }
57
58        @WebMethod(operationName = "getRoles")
59        public List<RoleWS> getRoles() {...12 lines }
71
72        @WebMethod(operationName = "createIoTDevice")
73        public void createIoTDevice(@WebParam(name = "deviceWS") IotDeviceWS deviceWS) {...3 lines }
76
77        @WebMethod(operationName = "getAllDevices")
78        public List<IotDeviceWS> getAllDevices() {...12 lines }
90
91        @WebMethod(operationName = "createLightData")
92        public void createLightData(@WebParam(name = "LightDataWS") LightDataWS dataWS) {...3 lines }
95
96        @WebMethod(operationName = "getLightData")
97        public List<LightDataWS> getLightData() {...12 lines }
109
110       @WebMethod(operationName = "createDht11Data")
111       public void createDht11Data(@WebParam(name = "Dht11DataWS") Dht11DataWS dataWS) {...3 lines }
114
115       @WebMethod(operationName = "getDht11Data")
116       public List<Dht11DataWS> getDht11Data() {...12 lines }
```

Slika 36: Implementacija - SOAP web servis

Drugi servis Sustava za upravljanje IoT uređajima je REST servis naziva *UserAuthentificationRest*. Primarna zadaća servisa je izvršavanje svih akcija koje su povezane s korisnikom (engl. User Management). Servis nudi korisnima mogućnost autentifikacije korisnika koja se koristi prilikom pristupanja web aplikacijama.

Servis je izrađen korištenjem JAX-RS specifikacijom za RESTful web servise. Kao i kod SOAP servisa, izrada web servisa je bitno olakšana korištenjem anotacija dostupnih pomoću specifikacije. Glavne korištene anotacije: @Path, @Context, @GET, @DELETE, @PUT, @POST, @Consumes i @Produces. Dio implementacije opisanog REST servisa prikazan je na slici broj 37.

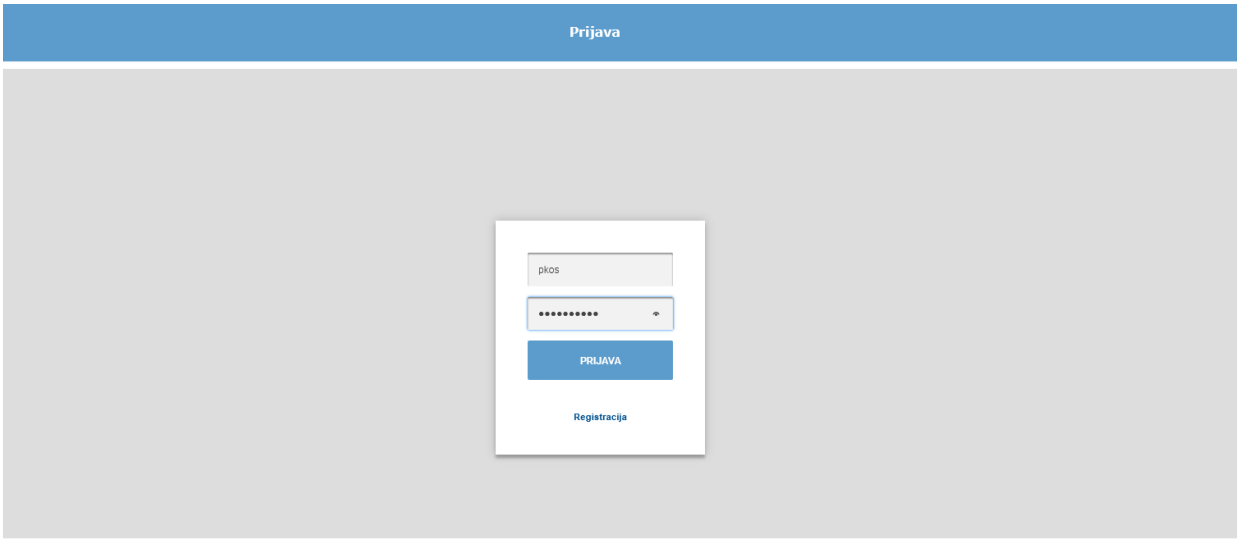
```
32   @Path("users")
33   public class UserAuthenticationRest {
34
35       @Context
36       private UriInfo context;
37
38       @GET
39       @Produces(MediaType.APPLICATION_JSON)
40       public String getJson() {...23 lines }
63
64       @POST
65       @Consumes(MediaType.APPLICATION_JSON)
66       @Produces(MediaType.APPLICATION_JSON)
67       public String postJson(String data) {...21 lines }
88
89       @Path("{id}")
90       @DELETE
91       @Produces(MediaType.APPLICATION_JSON)
92       public String deleteJsonId(@PathParam("id") String username) {...28 lines }
120
121      @Path("{id}")
122      @GET
123      @Produces(MediaType.APPLICATION_JSON)
124      public String getJsonId(@PathParam("id") String username) {...26 lines }
150
151      @Path("{id}")
152      @PUT
153      @Consumes(MediaType.APPLICATION_JSON)
154      @Produces(MediaType.APPLICATION_JSON)
155      public String putJsonId(@PathParam("id") String username, String data) {...44 lines }
199
```

Slika 37: Implementacija - REST web servis

## 5.5. Prikaz aplikacije

Sustav za upravljanjem IoT uređajima sastoji se od dvije aplikacije s grafičkim sučeljem. Glavna zadaća aplikacija je prijenos bitnih informacija korisnicima te mogućnost upravljanja IoT uređajima.

Prva aplikacija s grafičkim sučeljem, naziva Korisnički portal, orijentirana je korisnicima. Kako bi korisnik pristupio aplikaciji mora se prijaviti pomoću ranije kreiranog korisničkog računa. U slučaju da korisnik nema aktivan korisnički račun može ga kreirati putem Korisničkog portala. Slika 38 prikazuje formu za prijavu korisnika u Korisnički portal.




Slika 38: Prikaz aplikacije - Prijava u Korisnički portal

Nakon uspješne prijave korisniku se otvara početna stranica. Na početnoj stranici dostupne su glavne informacije o autoru cijelog sustava i popratnog rada. Slika 39 prikazuje početnu stranicu Korisničkog portala.

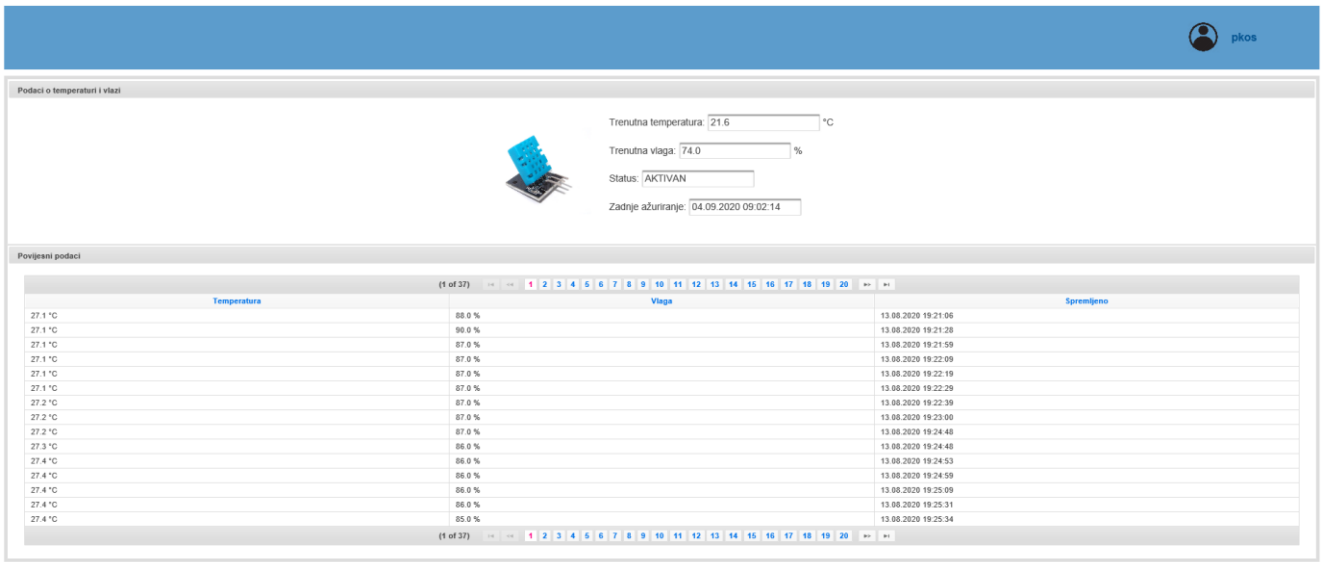



Slika 39: Prikaz aplikacije - Početna stranica Korisničkog portala

Korisnik jednom kad se prijavi ima mogućnost uređivanja svojeg profila. Svi korisnički podaci su podložni izmjenama osim korisničkog nadimka koji predstavlja jedinstvenu oznaku korisnika.

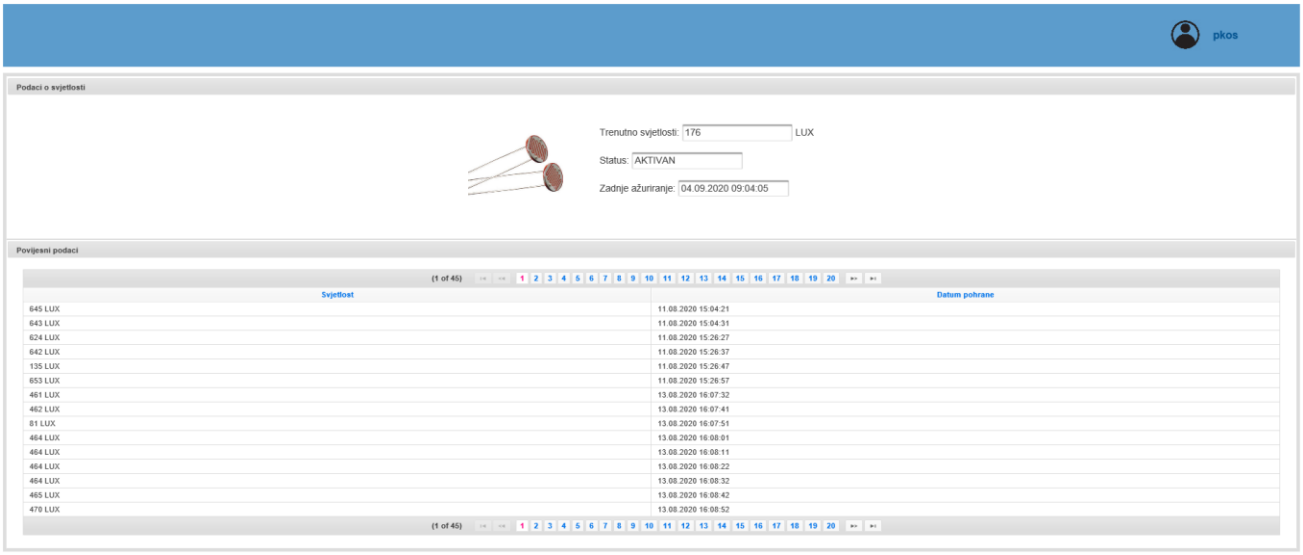
Prijavljeni korisnik na raspolaganju ima tri odvojene stranice koje prikazuju podatke prikupljene IoT uređajem.

vPrva web stranica, naziva Podaci o temperaturi i vlazi, prikazuje aktualne podatke prikupljene s DHT11 senzora. Opisa podataka o temperaturi i vlazi, prikaz je trenutni status uređaja i vrijeme zadnjeg ažuriranja podataka. Na dnu stranice nalazi se tablica s povijesnim podacima



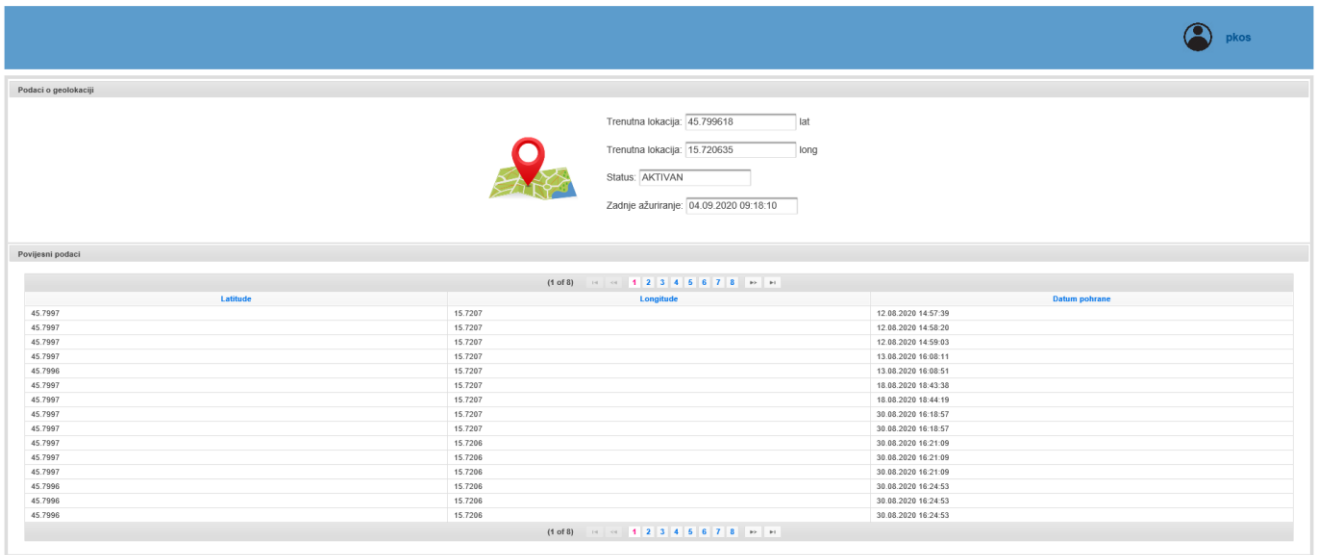

Slika 4010: Prikaz aplikacije - Podaci o temperaturi i vlazi

Slično prvoj stranici druga web stranica, naziva Podaci o svjetlosti, prikazuje podatke dobivene preko otpornika za svjetlo. Dobiveni podaci su obrađeni i prikazani korisniku u čitljivom obliku. Kao i kod prethodne stranice, korisniku je dostupan uvid u status, vrijeme ažuriranje te povijesne podatke senzora.




Slika 41: Prikaz aplikacije - Podaci o svjetlosti

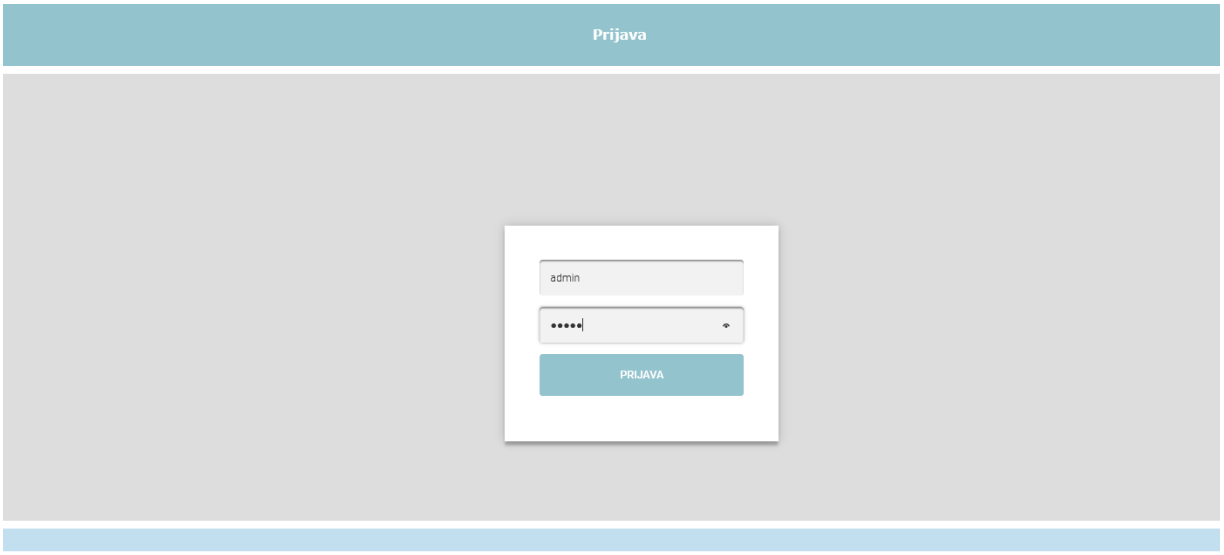
Posljednja web stranica unutar Korisničkog portala daje korisniku uvid u geolokacijske podatke. Stranica naziva podaci o geolokaciji prikazuje geografsku širinu i dužinu na kojoj se nalazi uređaj kao i status uređaja te vrijeme zadnjeg ažuriranja. U podnožju stranice nalazi se tablica s povijesnim podacima izmjerenim pomoću geolokacijskog uređaja




Slika 42: Prikaz aplikacije - Podaci o geolokaciji

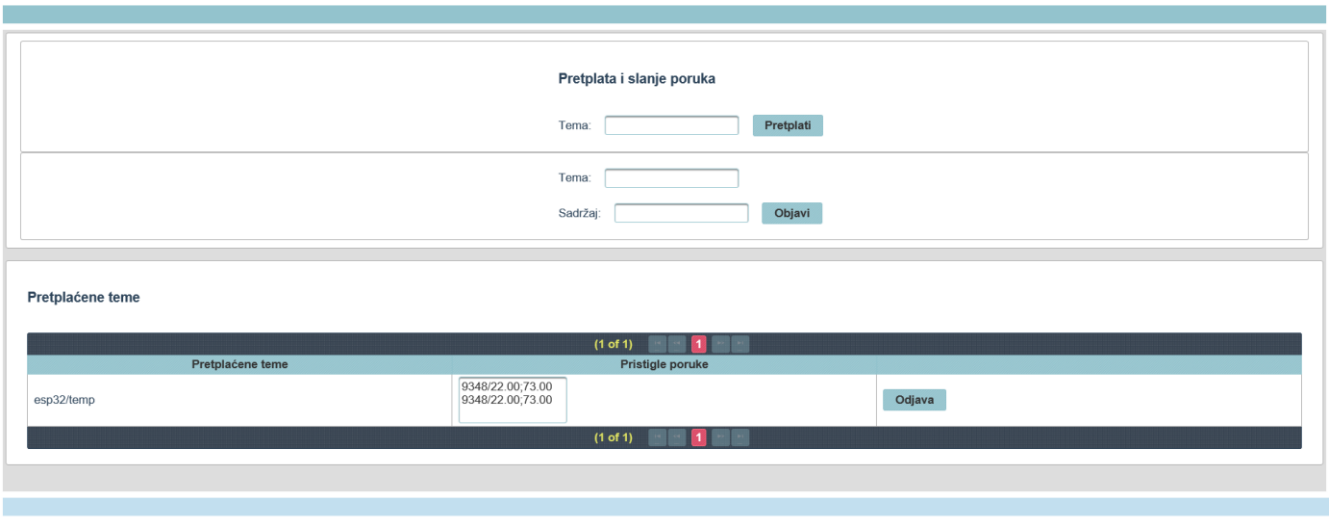
Druga aplikacija namijenjena je korisnicima s ulogom administratora. Aplikacija se naziva Admin aplikacija te pruža mogućnost upravljanja svim dostupnim IoT uređajima kao i uvid u sve akcije provedene unutar sustava te cjelokupni nadzor sustava. Korisnik s ulogom administratora ima mogućnost korištenja i Korisničkog portala.

Slično kao i u Korisničkom portalu, korisnik administrator se mora prijaviti u sustav kako bi ga mogao korisnici. Administratori nemaju mogućnost registracije, već im račune kreira razvojni programer zadužen za održavanje sustava. Slika 43 prikazuje formu za prijavu administratora.




Slika 43: Prikaz aplikacije - Prijava u Admin aplikaciju

Nakon uspješne prijave administratoru se otvara stranica MQTT konzola koja omogućava ručno upravljanje MQTT pretplatama. Administrator se putem konzole može pretplatiti na različite teme kao i slati poruke u bilo koju temu. Također, prikazane su sve teme na koje je pretplaćen kao i poruke koje su pristigle u odgovarajuće teme. Opisana web stranica vrlo je korisna za ručnu provjeru rada IoT uređaja kao i kontrolu rada cijelog sustava. Opisana web stranica prikaza je na slici 44.



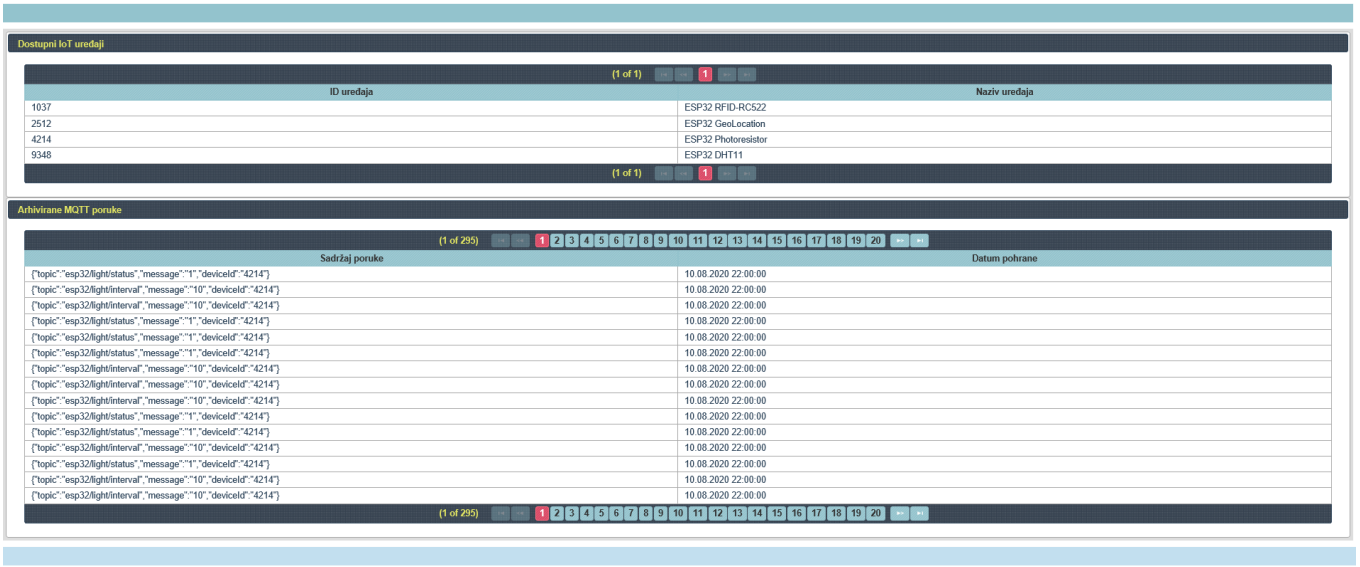

Slika 44: Prikaz aplikacije - MQTT konzola

Stranica Upravljanje IoT uređajima predstavlja glavnu stranicu unutar Admin aplikacije. Pomoću nje administrator ima mogućnost upravljanja svakim posebnim uređajem, ali i slanja poruka prema svim uređajima u isto vrijeme. Administrator svakom IoT uređaju može promijeniti interval objave podataka te zaustaviti odnosno ponovno pokrenuti uređaj. Opisana stanica nalazi se na slici 45.



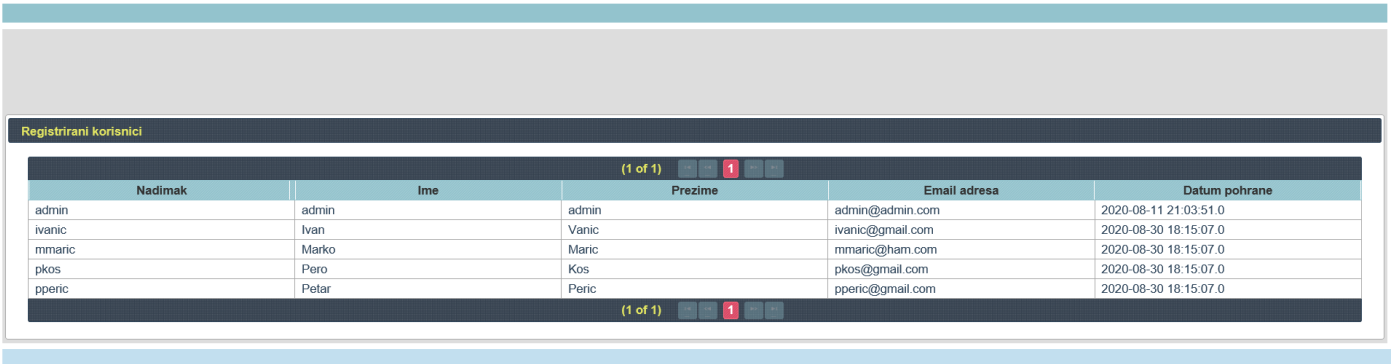

Slika 45: Prikaz aplikacije - Upravljanje IoT uređajima

Osim mogućnosti upravljanja IoT uređajima korištenjem MQTT poruka, administrator ima uvid u registrirane IoT uređaje te povijesni prikaz MQTT poruka zaprimljenih unutar cijelog sustava. Slika 46 prikazuje opisanu stranicu.



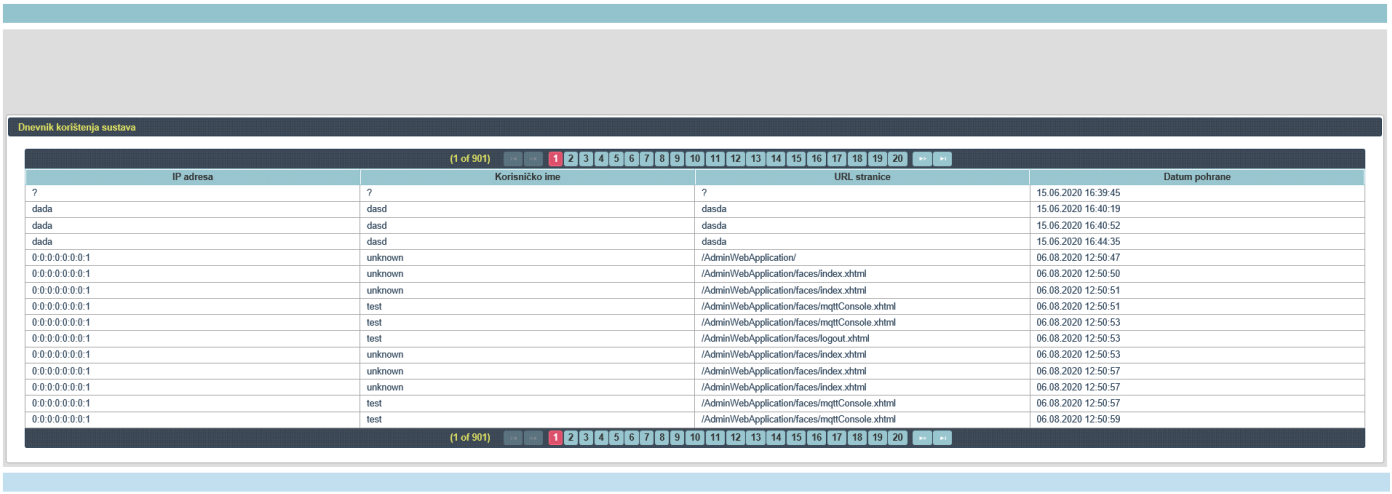

Slika 46: Prikaz aplikacije - Registrirani IoT uređaji i pristigle MQTT poruke

Također, administrator ima uvid u sve registrirane korisnike kao i njihove podatke. Pregled korisnika prikazan je na slici 47.




Slika 47: Prikaz aplikacije - Pregled registriranih korisnika

Zadnja stranica dostupna administratoru je Dnevnik. Pomoću dnevnika administrator prati rad svih korisnika unutar sustava. Korištenjem dnevnika i spremanjem podatak unutar sustava postiže se visoki stupanj sigurnosti. Prikaz Dnevnika nalazi se na slici 48.




Slika 48: Prikaz aplikacije - Dnevnički zapisi sustava

# 6. Zaključak

Temeljem teorijskog opisa IoT koncepta i MQTT poruka dolazimo do sljedećeg zaključka. IoT koncept veoma je popularan te je zadnjih nekoliko godina doživio ogroman rast i očekuje se da se takav trend rasta nastavi. IoT uređaji i senzori, iako su veoma korisni i imaju široku primjenu, ne čine potpuno funkcionalan sustav već samo zaseban dio. Kako bi se podaci dobiveni s IoT uređaja integrirali u sustav potreban je određen protokol. Unutar rada opisano je više protokola koji su pogodni prilikom korištenja IoT uređaja. Kao najrelevantniji protokol odabran je MQTT. Kombinacijom IoT uređaja i MQTT protokola podaci izmjereni putem senzora, bez gubitaka i u najmanjem mogućem vremenu, dolaze u nadzorni sustav gdje se dalje obrađuju.

Unutar rada predstavljene su i analizirane različite implementacije MQTT posrednika instaliranog na različitim platformama. Temeljem analize zaključeno je da posrednici na Windows operacijskom sustavu imaju vrlo slično vrijeme odaziva, dok posrednik na Raspberry Pi uređaju u usporedbi s Windows posrednikom ima znatno lošije rezultate. Prilikom kreiranja složenog projekta koji svoj rad temelji na niskom vremenu odaziva i visokoj stopi sigurnosti preporučuje se korištenje MQTT posrednika na Windows operacijskom sustavu. Kod manjih projekta nekomercijalne svrhe Raspberry Pi je dobar odabir platforme za instalaciju posrednika.

Osim protokola predstavljeni su web servisi kao jedna od mogućih metoda komunikacije između odvojenih komponenata sustava. SOAP i REST web servisi su detaljno analizirani te usporednom analizom pokazano je da se za razvoj novih servisa preporučuje korištenje REST servisa zbog boljeg vremena odaziva i lakše integracije. S druge strane, ukoliko se želi implementirati sustav s visokom razinom sigurnosti preporučuje se korištenje SOAP servisa.

Praktičan dio rada sastoji se od više komponenti koje čine složeni sustav pod nazivom Sustav za upravljanje IoT uređajima. Sustav kao što mu ime govori pruža korisnicima mogućnost upravljanje IoT uređajima te uvid u očitane podatke sa senzora. Sastoji se od više međusobno povezanih aplikacija, IoT uređaja, web servisa, baze podataka i MQTT posrednika instaliranog na Raspberry Pi uređaju. Svrha praktičnog dijela bila je prikazati jednu od mogućih implementacija IoT uređaja u složeni sustav te upravljanje uređajima putem MQTT protokola. Iako sustav nije namijenjen za komercijalne svrhe svojom arhitekturom i načinom implementacije predstavlja dobru praksu implementacije i upravljanja IoT uređaja.

# Popis literature


[1] D. Slama, F. Puhlmann, J. Morrish, and R. M. Bhatnagar, *Enterprise IoT : Strategies & Best Pratices for Connected Products & Services*. 2015.

[2] M. H. Asghar, A. Negi and N. Mohammadzadeh, "Principle application and vision in Internet of Things (IoT)," International Conference on Computing, Communication & Automation, Noida, 2015, pp. 427-431, doi: 10.1109/CCAA.2015.7148413.

[3] F. Meneghello, M. Calore, D. Zucchetto, M. Polese, and A. Zanella, "IoT: Internet of Threats? A Survey of Practical Security Vulnerabilities in Real IoT Devices," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8182–8201, 2019.

[4] J. Guth, U. Breitenbucher, M. Falkenthal, F. Leymann, and L. Reinfurt, "Comparison of IoT platform architectures: A field study based on a reference architecture," *2016 Cloudification Internet Things, CIoT 2016*, 2017.

[5] N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," 2017 IEEE International Systems Engineering Symposium (ISSE), Vienna, 2017, pp. 1-7, doi: 10.1109/SysEng.2017.8088251.

[6] S. Raza, H. Shafagh, K. Hewage, R. Hummen, and T. Voigt, "Lithe: Lightweight secure CoAP for the internet of things," *IEEE Sens. J.*, vol. 13, no. 10, pp. 3711–3720, 2013.

[7] Y. Jie, J. Y. Pei, L. Jun, G. Yun, and X. Wei, "Smart home system based on IOT technologies," *Proc. - 2013 Int. Conf. Comput. Inf. Sci. ICCIS 2013*, pp. 1789–1791, 2013.

[8] L. Catarinucci *et al.*, "An IoT-Aware Architecture for Smart Healthcare Systems," *IEEE Internet Things J.*, vol. 2, no. 6, pp. 515–526, 2015.

[9] H. Arasteh et al., "Iot-based smart cities: A survey," 2016 IEEE 16th International Conference on Environment and Electrical Engineering (EEEIC), Florence, 2016, pp. 1-6, doi: 10.1109/EEEIC.2016.7555867.

[10] "Internet of Things units installed base by category 2014-2020 | Statista." [Na mrežo]. Dostupno: https://www.statista.com/statistics/370350/internet-of-things-installed-base-by-category/. [Pristupljeno: 14.07.2020].

[11] S. K. Lee, M. Bae, and H. Kim, "Future of IoT networks: A survey," *Appl. Sci.*, vol. 7, no. 10, pp. 1–25, 2017.

[12] "MQTT Version 5.0." [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html. [Accessed: 10-May-2020].

[13] Gaston C. Hillar. MQTT Essentials - A Lightweight IoT Protocol. January 2017.

[14] R. Banno, J. Sun, M. Fujita, S. Takeuchi, and K. Shudo, “Dissemination of edge-heavy data on heterogeneous MQTT brokers,” *Proc. 2017 IEEE 6th Int. Conf. Cloud Networking, CloudNet 2017*, 2017.

[15] D. Thangavel, X. Ma, A. Valera, H. X. Tan, and C. K. Y. Tan, “Performance evaluation of MQTT and CoAP via a common middleware,” *IEEE ISSNIP 2014 - 2014 IEEE 9th Int. Conf. Intell. Sensors, Sens. Networks Inf. Process. Conf. Proc.*, no. April, pp. 21–24, 2014.

[16] S. Lee, H. Kim, D. K. Hong, and H. Ju, “Correlation analysis of MQTT loss and delay according to QoS level,” *Int. Conf. Inf. Netw.*, pp. 714–717, 2013.

[17] “Eclipse Mosquitto.” [Online]. Available: https://mosquitto.org/. [Accessed: 14-Jul-2020].

[18] “ActiveMQ.” [Online]. Available: https://activemq.apache.org/. [Accessed: 14-Jul-2020].

[19] “Moquette Broker.” [Online]. Available: https://moquette-io.github.io/moquette/. [Accessed: 14-Jul-2020].

[20] “Messaging that just works — RabbitMQ.” [Online]. Available: https://www.rabbitmq.com/. [Accessed: 14-Jul-2020].

[21] K. Grgić, I. Špeh and I. Heđi, "A web-based IoT solution for monitoring data using MQTT protocol," 2016 International Conference on Smart Systems and Technologies (SST), Osijek, 2016, pp. 249-253, doi: 10.1109/SST.2016.7765668.

[22] P. Dhar and P. Gupta, "Intelligent parking Cloud services based on IoT using MQTT protocol," 2016 International Conference on Automatic Control and Dynamic Optimization Techniques (ICACDOT), Pune, 2016, pp. 30-34, doi: 10.1109/ICACDOT.2016.7877546.

[23] “Overview of the JMS API.” [Online]. Available: https://javaee.github.io/tutorial/jms-concepts001.html. [Accessed: 19-Aug-2020].

[24] Tarak Modi (2002). Practical Java Message Service An Introduction to concepts and a guide to developing applications

[25] D. Chappell and T. Jewell, Java Web Services. 2013.

[26] “SOAP Specifications.” [Online]. Available: https://www.w3.org/TR/soap/. [Accessed: 19-Aug-2020].

[27] “The Java EE 5 Tutorial” [Online]. Available: https://docs.oracle.com/cd/E19879-01/819-3669/bnbhj/index.html. [Accessed: 19-Aug-2020].

[28] Graham, Steve & Davis, Doug & Simeonov, Simeon & Daniels, Glen & Brittenham, Peter & Nakamura, Yuichi & Fremantle, Paul & König, Dieter & Zentner, Claudia. (2002). Building Web services with Java: Making sense of XML, SOAP, WSDL, and UDDI.

[29] Halili, Festim & Ramadani, Erenis. (2018). Web Services: A Comparison of Soap and Rest Services. Modern Applied Science. 12. 175. 10.5539/mas.v12n3p175.

[30] “Building Web Services with JAX-WS [Online]. Available: https://docs.oracle.com/javaee/6/tutorial/doc/bnayl.html. [Accessed: 19-Aug-2020].

[31] J. Sandoval, RESTful Java Web Services. 2009.

[32] B. Burke, RESTful Java with JAX-RS 2.0 - Designing and Developing Distributed Web Services. 2013.

[33] “Creating a RESTful Root Resource Class.” [Online]. Available: https://javaee.github.io/tutorial/jaxrs002.html. [Accessed: 19-Aug-2020].

[34] Wagh, Dr. K & Thool, Ravindra. (2012). A Comparative study of SOAP vs REST web services provisioning techniques for mobile host. Journal of Information Engineering and Applications. 2. 12-16.

[35] K. P. Pavan, A. Sanjay, and P. Zornitza, “Comparing Performance of Web Service Interaction Styles : SOAP vs . REST,” Proc. Conf. Inf. Syst. Appl. Res., pp. 1–24, 2012.

[36] S. Bipasha Biswas and M. Tariq Iqbal, "Solar Water Pumping System Control Using a Low Cost ESP32 Microcontroller," 2018 IEEE Canadian Conference on Electrical & Computer Engineering (CCECE), Quebec City, QC, 2018, pp. 1-5, doi: 10.1109/CCECE.2018.8447749.

# Popis slika

Slika 1: Arhitektura IoT sustava [4] ..... 3
Slika 2: Prikaz IoT protokola [5] ..... 5
Slika 3: Područja primjene IoT-a [2] ..... 7
Slika 4: Rast i područje primjene IoT-a [10] ..... 9
Slika 5: Proces razmjene poruka pomoću MQTT protokola ..... 11
Slika 6: Uspostava veze između klijenta i poslužitelja [13] ..... 12
Slika 7: Prikaz QoS 0 razine ..... 13
Slika 8: Prikaz QoS 1 razine ..... 14
Slika 9: Prikaz QoS 2 razine ..... 14
Slika 10: Rezultati usporedne analize posrednika (autorski rad) ..... 17
Slika 11: Arhitektura sustava temeljenog na MQTT porukama [21] ..... 19
Slika 12: Arhitektura sustava za pametno parkiranje automobila [22] ..... 21
Slika 13: Nadopunjeni rezultati usporedne analize posrednika (autorski rad) ..... 22
Slika 14: P2P stil [23] ..... 24
Slika 15: Objavi / Pretplati stil [23] ..... 24
Slika 16: SOAP poruka [27] ..... 27
Slika 17: Proširena SOAP poruka [27] ..... 28
Slika 18: SOAP vs REST - Žičana veza [35] ..... 33
Slika 19: SOAP vs REST - Bežična veza [35] ..... 34
Slika 20: Komponente Sustava za upravljanje IoT uređajima (autorski rad) ..... 36
Slika 21: Dijagram slučaja korištenja sustava ..... 39
Slika 22: Dijagram slijeda aktivnosti - Prijava u sustav ..... 40
Slika 23: Dijagram slijeda aktivnosti - Prikaz podataka očitanih s IoT uređaja ..... 41
Slika 24: Dijagram slijeda aktivnosti - Upravljanje IoT uređajem ..... 42
Slika 25: Dijagram klasa - DB aplikacija ..... 43
Slika 26: Dijagram klasa - EJB modul ..... 44
Slika 27: ERA model sustava ..... 45
Slika 28: Postavi metoda ..... 47
Slika 29: Izvodi metoda ..... 48
Slika 30: Shema spajanja - ESP32 i DHT11 senzor ..... 49
Slika 31: Shema spajanja - ESP32 i svjetlosni senzor ..... 49
Slika 32: Shema spajanja - ESP32 i RFID-RC522 senzor ..... 50
Slika 33: Način komunikacije IoT uređaja i MQTT posrednika ..... 51
Slika 34: Način komunikacije EJB modula, MQTT posrednika i aplikacija ..... 52
Slika 35: Hibernate konfiguracijska datoteka ..... 53
Slika 36: Implementacija - SOAP web servis ..... 54
Slika 37: Implementacija - REST web servis ..... 55

Slika 38: Prikaz aplikacije - Prijava u Korisnički portal .....56
Slika 39: Prikaz aplikacije - Početna stranica Korisničkog portala.....57
Slika 40: Prikaz aplikacije - Podaci o temperaturi i vlazi.....57
Slika 41: Prikaz aplikacije - Podaci o svjetlosti.....58
Slika 42: Prikaz aplikacije - Podaci o geolokaciji.....58
Slika 43: Prikaz aplikacije - Prijava u Admin aplikaciju.....59
Slika 44: Prikaz aplikacije - MQTT konzola.....60
Slika 45: Prikaz aplikacije - Upravljanje IoT uređajima.....60
Slika 46: Prikaz aplikacije - Registrirani IoT uređaji i pristigle MQTT poruke.....61
Slika 47: Prikaz aplikacije - Pregled registriranih korisnika .....61
Slika 48: Prikaz aplikacije - Dnevnički zapisi sustava.....62

# Popis tablica

Tablica 1: Analiza IoT protokola [5]..... 6
Tablica 2: Analiza brzine prijenos podataka kod različitih QoS konfiguracija [16].....15
Tablica 3: JAX-RS anotacije [33].....30
Tablica 4: Usporedba karakteristika SOAP i REST web servisa [34] .....32