UNIVERZITET U NIŠU
ELEKTRONSKI FAKULTET

Platforma za Internet stvari bazirana na arhitekturi mikroservisa

Master rad
Studijski program: Računarstvo i informatika
Modul: Inženjerstvo podataka

Student:

Danilo Veljović, br. ind. 1120

Mentor:

Doc. dr. Aleksandar Stanimirović

Niš, septembar 2021. god.

Univerzitet u Nišu
Elektronski fakultet

Platforma za Internet stvari bazirana na arhitekturi mikroservisa

Internet of Things platform based on microservice architecture

Master rad
Studijski program: Računarstvo i informatika
Modul: Inženjerstvo podataka

Student: Danilo Veljović

Mentor: Doc. dr. Aleksandar Stanimirović

Zadatak: *Proučiti i prikazati moguće arhitekture platformi za Internet stvari. Detaljno izučiti platformu zasnovanu na arhitekturi mikroservisa. Implementirati primer platforme zasnovan na arhitekturi mikroservisa.*

Datum prijave rada: 15.07.2021

Datum predaje rada:

Datum odbrane rada:

Komisija za ocenu i odbranu:

1. Doc dr Aleksandar Stanimirović, Predsednik Komisije

2. <Prof. / Doc dr Ime Prezime>, Član

3. <Prof. / Doc dr Ime Prezime>, Član

PLATFORMA ZA INTERNET STVARI BAZIRANA NA ARHITEKTURI MIKROSERVISA

Sažetak

Internet stvari je pojam koji poslednjih godina sve više dobija na važnosti. Usled razvoja "*pametnih*" senzora moguće je pribaviti sve detaljnija merenja i slati ih na obradu do nekog "edge" uređaja. *Edge* uređaji imaju sve veću moć obrade i mogu da u realnom vremenu skladište i analiziraju primljene vrednosti, kao i da vrše aktuaciju. U ovom radu, predloženo je rešenje koje objedinjuje sve ove funkcionalnosti, u vidu *cloud platforme* za internet stvari. Cloud platforme su sve popularnije, jer omogućavaju pristup podacima sa bilo kog uređaja koji može da se konektuje na internet. Proučavane su različite arhitekture cloud platformi. Neke od arhitektura koje su proučene su monolitna arhitektura i mikroservisna arhitektura. Sagledane su prednosti i mane svake arhitekture, kao i neki primeri korišćenja. Kao fokus ovog rada, posebno je proučavana arhitektura mikroservisa. Arhitektura mikroservisa je sve popularnija, naročito za web aplikacije. Omogućava jednostavno skaliranje, postavljanje na server i razdvajanje servisa. Ovim se sistem struktuira u celine koje mogu međusobno nezavisno da funkcionišu. U okviru arhitekture mikroservisa, proučavano je kako se mogu izdvajati nezavisne funkcionalnosti i kako se mogu dodeljivati posebnim servisima. Poseban akcenat je dat na razmatranje, koji servisi bi trebalo da se izvršavaju na edge delu sistema, a koji na cloud delu sistema.

U praktičnom delu rada implementiran je prototip platforme za internet stvari baziran na arhitekturi mikroservisa. Aplikacija je kreirana za svrhe agrikulture, odnosno navodnjavanja zemljišta i predikcije kvaliteta zemlje. Razvijeni prototip sistema se sastoji od servisa koji se bave skladištenjem podataka, analizom podataka, izvršavanjem upita i aktuacijom. Na kraju rada se disktuju predlozi za poboljšanje sistema, kao i još neki mogući načini primene.

Ključne reči: arhitektura mikroservisa, Internet stvari, platforma, cloud, edge.

INTERNET OF THINGS PLATFORM BASED ON MICROSERVICE ARCHITECTURE

ABSTRACT

Internet of things is becoming an increasingly relevant topic in the recent years. Because of the development of smart sensors, it is possible to obtain very frequent and detailed measurements and to send them to and edge node to be processed. The processing power of edge devices is increasing, and they can store and analyze the received data in real-time, as well as do an actuation. In this thesis, a solution is suggested to unite all of these functionalities in the form of a cloud platform. Cloud platforms are all the more popular solution these days, because they allow access to the data from any device that can be connected to the internet. Different kinds of cloud architectures were explored. Some of those are monolithic architecture and microservice architecture. Benefits and downsides to each architecture were studied, as well as some examples of use. The focus of this work is the microservice architecture. Microservice architecture is ever more popular architecture solution especially for the web applications. They allow simple scaling of services, independent deployment and loosely coupled services. This way the system is structured as a set of independent parts that can work together, without being coupled. The way in which the functionalities of the system can be discerned and how can they be assigned to services is also explored. A special accent is placed on the investigation of which services should be deployed to an edge device, and which should be deployed on the cloud.

The second part of the thesis contains an implementation of a prototype of the cloud platform based on the microservice architecture. Application is created for the purposes of agriculture, irrigation of the land and prediction of quality of the soil. The developed protype consists of services that store the data, analyze it, query it and send actuation requests. Finally, in the final section of the thesis, suggestions are given on how to improve the platform and few new ways on where it could also be used.

Keywords: microservice architecture, internet of things, platform, cloud, edge.

SADRŽAJ

1. UVOD

Internet stvari je pojam koji poslednjih godina sve više dobija na važnosti. Zbog velike količine podataka koju je danas moguće preneti internetom, moguće je u realnom vremenu pratiti stanje bolnica, nuklearnih elektrana, poljoprivrednih imanja itd. Usled razvoja "pametnih" senzora, moguće je prikupiti sve detaljnija merenja u sve kraćim vremenskim intervalima. Danas senzori mogu da vrše merenja u intervalima od par milisekundi, što može da rezultira u po više stotina merenja u sekundi. Ovako veliku količinu podataka, potrebno je dostaviti do edge ili cloud uređaja. Na edge ili cloud sloju se prikupljeni podaci perzistiraju, analiziraju, i ako su merenja takva da se neki uslov ispunio, vrši se i aktuacija određenih uređaja.

Funkcionalnosti poput skladištenja primljenih podataka, zatim njihova analiza, i aktuacija uređaja predstavljaju srž svakog sistema za internet stvari. Sistemi za internet stvari moraju da objedinjuju sve ove funkcionalnosti i zbog toga se vrlo često i nazivaju platformom za internet stvari. Platforme su popularno rešenje jer omogućavaju pristup podacima sa bilo kog uređaja koji se može povezati na internet. Ovim je omogućena cross-platform podrška, koju svi korisnici očekuju. Potrebno je omogućiti da platforma funkcioniše, čak i ako je broj podataka koji treba da se obradi reda veličine nekoliko desetina gigabajta ili par desetina terabajta. Za sisteme koji su projektovani u monolitnoj arhitekturi, ovolika količina podataka može predstavljati problem.

Monolitni sistemi teže skaliraju. Ako odjednom u sistem krene da pristiže neuobičajeno velika količina podataka, potrebno je da se cela aplikacija skalira. Nije moguće samo imati redundantne instance jednog dela sistema. U monolitnim sistemima se teže dodaju nove funkcionalnosti. Vrlo često, usled kvara na jednom delu sistema, cela aplikacija prestaje da funkcioniše. Ovo može da izazove velike probleme, ili čak nedetektovane događaje na koje je bilo potrebno reagovati.

Kao predloženo rešenje problema monolitnih sistema data je arhitektura mikroservisa. Arhitektura mikroservisa je standard u projektovanju web aplikacija današnjice. Omogućava podelu sistema na nezavisne servise, omogućava nezavisno postavljanje na server tih servisa i njihovo jednostavno skaliranje. Arhitektura mikroservisa uvodi i određene probleme poput intenzivnije komunikacije između servisa, težeg održavanja konzistentnosti podataka u sistemu itd.

Arhitektura mikroservisa predstavlja rešenje mnogih problema kod aplikacija internet stvari. Vrlo često je potrebno skladištiti ili analizitati veliku količinu podataka. Ako bi postojao nezavistan servis koji skladišti podatake, on bi se mogao skalirati u toj meri, da sistem može da nastavi da normalno funkcioniše. Ako iz nekog razloga, servis za analitiku ne funkcioniše, to neće dovesti do pada celog sistema. Sistem razvijen na ovaj način je pouzdan, otporan na padove, ima nizak down time i skalabilan je.

U narednom poglavlju biće proučene različite arhitekture IoT platformi. Proučiće se prednosti i mane svake arhitekture. Poseban akcenat biće dat na arhitekturi mikroservisa, budući da je ona fokus ovog rada. U trećem poglavlju je opisana implementacija IoT platforme bazirane na arhitekturi mikroservisa. Četvrto poglavlje je diskusija o prednostima i manama ovog sistema, kao i o mogućnostima za poboljšanje implementiranog prototipa.

2. IOT SISTEMI

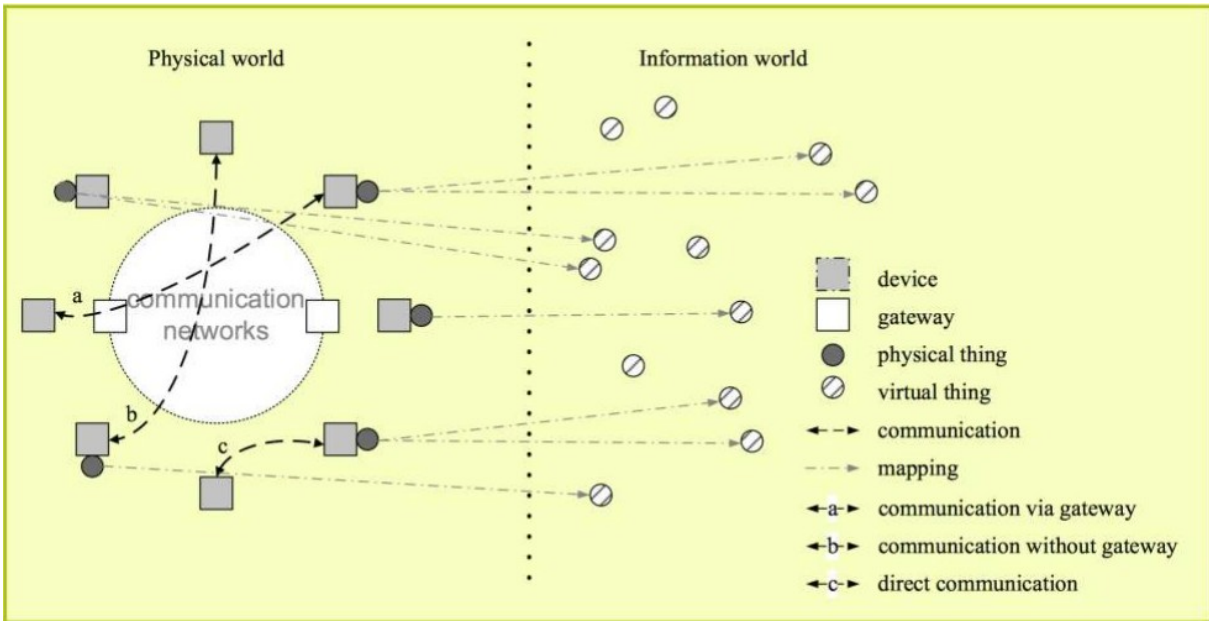
2.1.IoT sistemi

Internet stvari (engl. *Internet of things*) predstavlja mrežu povezanih *stvari* (engl. *things*). Stvari predstavljaju fizičke uređaje ili servise koji se izvršavaju na nekoj mašini. Stvari imaju ugrađene u sebe senzore kojima mogu prikupljati informacije iz svoje spoljašnjosti. Zatim tako prikupljene informacije mogu obrađivati ili slati drugim uređajima. [1]

Stvari se dele na:

- *Fizičke stvari*
- *Virtuelne stvari*

Fizičke stvari su hardverski uređaji koji imaju senzore, pomoću kojih detektuju stvari iz svoje okoline. Imaju sposobnost *aktuacije* tj. reagovanja na događaje i uticaja na svoju okolinu. Fizičke stvari mogu da detektovane podatke šalju drugim uređajima. Primeri fizičkih uređaja su industrijski roboti, senzori, Raspberry Pi itd. Virtuelne stvari postoje u vidu softvera. Virtuelnim stvarima se može pristupati, mogu se čuvati, i obrađivati. Primeri virtuelnih stvari su multimedijalni sadržaj i aplikativni softver.[2] Na slici 1 dat je prikaz jednog IoT sistema i interakcija između fizičkih i virutelnih stvari. Komunikacija između stvari se dešava preko interneta. Internet stvari obuhvata i vidove komunikacije poput machine-to-machine (srp. Mašina sa mašinom), bežične senzorske mreže, senzorske mreže, 2G/3G/4G/5G, GSM, GPRS, RFID, WI-FI, GPS itd. [3]



Slika 1: Ilustraciia virtuelnih i fizičkih stvari [2]

Tehnologije koje učestvuju u internetu stvari se mogu podeliti u tri grupe:

1. Tehnologije koje omogućavaju stvarima da prikupe informacije iz okoline – senzori
2. Tehnologije koje omogućavaju stvarima da obrade informacije iz okoline
3. Tehnologije koje poboljšavaju privatnost i sigurnost

Prve dve kategorije predstavljaju funkcionalne blokove bilo kog sistema za internet stvari. [3] Tu spadaju tehnologije koje omogućavaju uređajima da budu opaženi, i kontrolisani putem postojeće mrežne infrastrukture. Težnja je ostvariti što efikasniju, tačniju i nezavisniju komunikaciju između velikog broja različitih uređaja. Poslednjih godina, razvojem veštačke inteligencije, se teži da se čovek eliminiše iz ove komunikacije. Sistemi se projektuju tako da funkcionišu bez ljudskog uplitanja. Treća kategorija tehnologija dobija na važnosti otkako se sve više uređaja integriše u IoT sisteme. Potrebno je osigurati da komunikacija između IoT uređaja bude sigurna i bezbedna. Ne sme biti sigurnosnih propusta koje bi potencijalni napadači mogli da iskoriste. [3][1]

2.2. Karakteristike IoT sistema

Neke od osnovnih karakteristika IoT sistema su:

- **Povezanost:** Bilo koja stvar se može povezati u sistem bez obzira na njene hardverske mogućnosti ili softversku specifikaciju.
- **Heterogenost:** Uređaji u IoT sistemima su heterogeni. Uređaji imaju različite softverske I hardverske karakteristike. Koriste različite protokole za komunikaciju. Takve uređaje je potrebno objediniti u jedinstven sistem.
- **Dinamične promene:** Stanja uređaja se dinamično menjaju. Uređaji spavaju (engl. sleep) I bude se (engl. wake up), povezuju se na mreže. Menja im se lokacija, brzina (engl. velocity). Broj uređaja se takođe dinamički menja.
- **Ogromna razmera sistema:** Broj uređaja u IoT sistemima je od nekih 1000 uređaja u manjim sistemima, do 100.000 uređaja u velikim industrijskim sistemima. Potrebno je omogućiti da sistem funkcioniše pouzdano. Upravljanje podacima je posebno problematično u ovako velikim sistemima. Potrebno podatke struktuirati na pravi način tako da svaki od uređaja može da ih šalje, prima I obrađuje.
- **Bezbednost:** Sistem mora da se dizajnira tako da bude bezbedan. Mora se obezbediti bezbednost ličnih podataka I bezbednost sistema. Potrebno je primeniti sigurne protokole za komunikaciju, obezbediti svaki endpoint u sistemu I integritet postojećih podataka.
- **Servisi koji su okrenuti stvarima:** IoT sistemi mogu da obezbede servise koji su okrenuti stvarima. Ti servisi obezbeđuju potrebnu privatnost I semantičku konzistensnost između fizičkih I virtuelnih uređaja. Tehnologije koje podržavaju servise okrenute stvarima će morati da se menjaju kako bi obezbedili potrebni stepen privatnosti I pouzdanosti I kako bi se odgovorilo na ograničenja koje stvari imaju. [3]

Na slici 2 dat je prikaz svih tehnologija koje učestvuju u IoT sistemima. Karakteristike

Slika 2: Tehnologije koje učestvuju u internetu stvari

2.3. Arhitektura IoT sistema

Na slici 3 se nalazi prikaz tipične IoT arhitekture, tj elemenata koji su sastavni deo svakog IoT sistema.

Slojevi u strukturi IoT sistema su:

A) Sloj pametnih senzora/uređaja (engl. *smart device/sensor layer*) :

Najniži sloj se sastoji od "pametnih" stvari sa ugrađenim senzorima. Senzori omogućavaju povezivanje "pametnog" uređaja I okoline. Senzor detektuje fizičku pojavu I pretvaraju je u električni signal koji uređaj razume. Senzorom se detektuju I mere određene pojave od interesa u realnom vremenu. Postoje različite vrste senzora u zavisnosti od događaja koji se detektuje, kao I sa različitim stepenima osetljivosti I preciznosti u merenju. Postoje senzori za merenje temperature, vazdušnog pritiska, brzine, geografske širine I dužine itd. Senzori mogu imati određenu količinu memorije kojom mogu čuvati određenih broj najnovijih merenja. Informacije koje su dobijene sa senzora se obično transportuju putem mrežnog sloja do sloja za upravljanje servisima.[3][1]

B) Mrežni sloj (engl. *network layer*)

Senzori proizvode velike količine podataka. Prenos te količine podataka zahteva robustan I pouzdan žičani/bežični mehanizam koji može da podnese po nekoliko terabajta podataka u sekundi. Takođe je potrebno da se obezbedi komunikacija između senzora koji se nalaze na različitim mrežama. Kako bi se ovo ostvarilo mrežni sloj mora da bude u stanju da podrži veliki broj uređaja, da razume protokole koje ti uređaji koriste, I da rutira poruke na pravi način. [3]

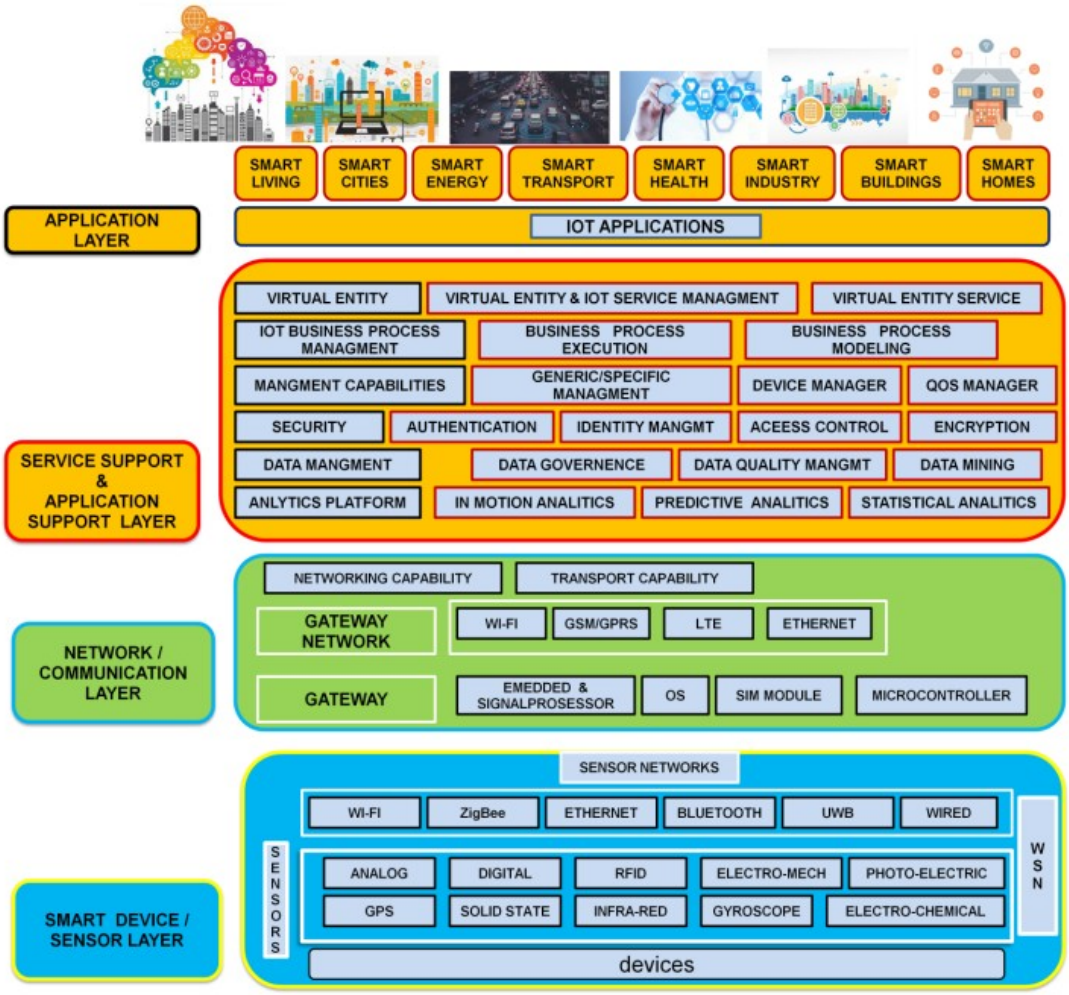
C) Sloj za upravljanje servisima (engl. *Management Service Layer*)

Ovaj sloj je zadužen za obradu informacija, analitiku, sigurnosne kontrole, modelovanje procesa I upravljanje uređajima. Jedan od najvažnijih uloga ovog sloja je pomoć pri donošenju odluka. Ovaj sloj prikuplja informacije, obrađuje ih u skladu sa određenim pravilima I skladišti ih nakon obrade. Takođe analizira pristigle podatke I detektuje trendove u podacima ili podatke/događaje koji su netipični. Na osnovu ovoga sistemu ili korisniku se pruža mogućnost da reaguje na određeni način. [3][1]

Ovaj sloj se takođe bavi I upravljanjem toka podataka. Pod upravljanjem podataka est podrazumeva pristup podacima, integracija I kontrola toka. Sloj za upravljanje servisima može da filtrira podatke koji idu ka aplikativnom sloju I time da smanje broj podataka koji će aplikativni sloj morati da obradi. Takođe ovaj sloj može da vrši prepakivanje podataka I njihovu agregaciju kako bi se samo najvažnije informacije prosleđivale. [3]

D) Aplikativni sloj

Na ovom sloju se nalaze aplikacije koje koriste primljene podatke. Neke od poznatijih primena IoT aplikacija su u domenu transporta, agrikulture, pametnih gradova, medicine, prodaje itd. [3][1]



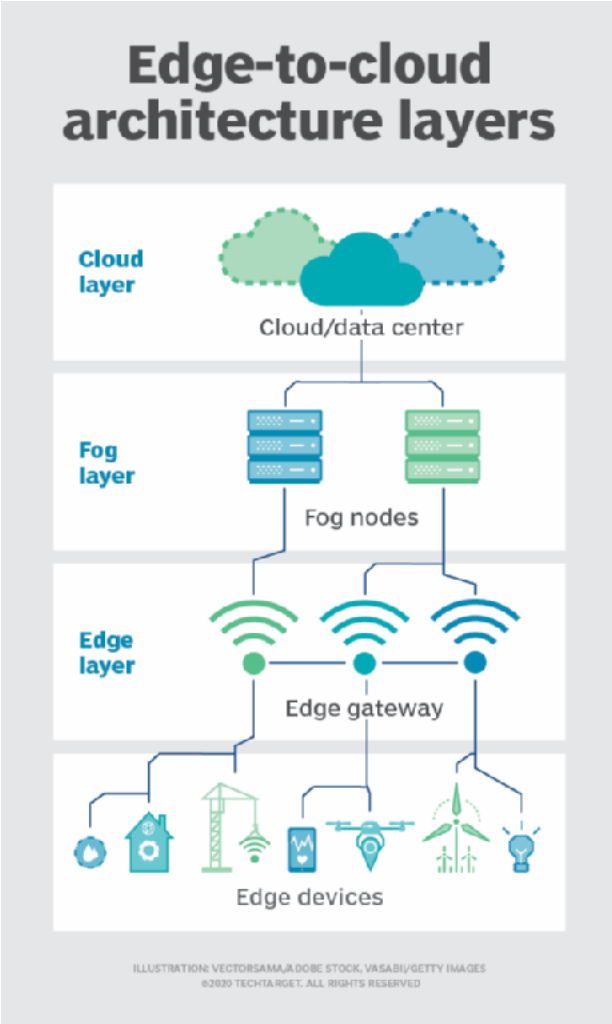
Slika 3: Arhitektura IoT sistema[3]

2.4. Topološka arhitektura IoT sistema

U današnjem svetu koncept IoT arhitekture je adaptiran tako da odgovara potrebama i mogućnostima savremenih računarskih sistema. Po blizini obrade podataka u odnosu na izvor i po procesnoj moći, savremeni IoT sistemi imaju tri sloja:

- Edge layer (srp. *Ivični sloj*)
- Fog sloj (srp. *Sloj magle*)
- Cloud sloj (srp. *Sloj oblaka*)

Na slici 4 je dat prikaz savremene IoT arhitekture i sve slojeve koje obuhvataju.



Slika 4: Savremena IoT arhitektura[4]

Kod ovih slojeva je slično to da prihvataju distribuiranu prirodu IoT sistema. Razlika između njih je u tome gde se nalaze u odnosu na izvor podataka i koliku procesnu moć imaju.

Edge sloj

Kod Edge sloja se računarski resursi koji služe za obradu i skladištenje informacija nalaze na izvoru ili neposredno blizu izvora podataka. U ovom slučaju se latencija u prenosu izmerenih podataka potpuno eliminiše. Ovo znači da se podaci mogu čitati i analizirati u realnom vremenu, tako da se sve promene mogu detektovati odmah i na njih se može adekvatno reagovati. Ono što je problem kod Edge sloja jeste ograničena procesna moć uređaja koji ga čine. Za razliku od uređaja na Fog i Cloud sloju, ovde se obično stavljaju uređaji ograničene procesne moći, što može da oteža obradu podataka. [4][1]

Jedan od slučajeva korišćenja edge sloja je kod vetroturbina. Blizu vetroturbina se može dodati par servera pomoću kojih bi se skupljali i analizirali podaci koje vetroturbina prikuplja. Još jedan od primera je železnička stanica. Na njoj se mogu postaviti serveri

skromnijih performansi, kojima bi se mogli prikupljati i analizirati razne informacije koje dolaze sa raznih senzora postavljenih na stanici. [4]

Fog sloj

Problem edge sloja je što su resursi ograničenih performansi ili previše razuđeni. Problem cloud sloja je u tome što je vrlo često predaleko od izvora informacija. Kako bi se premostio jaz koji postoji između cloud i edge sloja, osmišljena je ideja Fog sloja. Fog sloj približava moćne računarske resurse, slične kakve postoje na cloud sloju, izvoru podataka.

Tipičan primer korišćenja Fog sloja je u sistemima gde se generiše previše informacija za edge sloj, pa je potrebno obezbediti veću moć obrade podataka. Takvi su slučajevi "pametnih" zgrada , "pametnih" gradova itd. Ako se uzme primer pametnih gradova, tu se pomoću podataka sa senzora može pratiti sistem javnog prevoza, gradske usluge, komunalne usluge, pratiti trendovi u urbanom razvoju grada itd. Jedan edge sloj neće biti dovoljan da podnese toliko opterećenje. Zato se dodaje dovoljan broj Fog čvorova (engl. *node*) koji će pomoći u obradi svih pristiglih podataka. [4]

Cloud sloj

Cloud sloj se takođe naziva i računarstvo u oblaku. Cloud sloj je najveći od svih slojeva, ima najveću moć obrade i skalabilan je. Obično je Cloud sloj krajnji sloj jedne IoT arhitekture. Cloud provajderi obično pružaju i određene servise za IoT operacije, i time olakšavaju deployment, monitoring i skaliranje aplikacija i servisa koji se tamo izvršavaju. Cloud je na taj način postao omiljena platforma za IoT servise, jer je potrebna minimalna konfiguracija kako bi se neki servis podigao na server. Takođe ogromna procesna moć koju ima Cloud sloj garantuje da se može opslužiti bilo koja količina podataka. [4]

Mana Cloud sloja je ta što je potrebno određeno vreme dok se podaci transportuju od izvora do cloud-a. Ovo može da ima negativnog uticaja, ako je potrebno u realnom vremenu pratiti performanse sistema. [4]

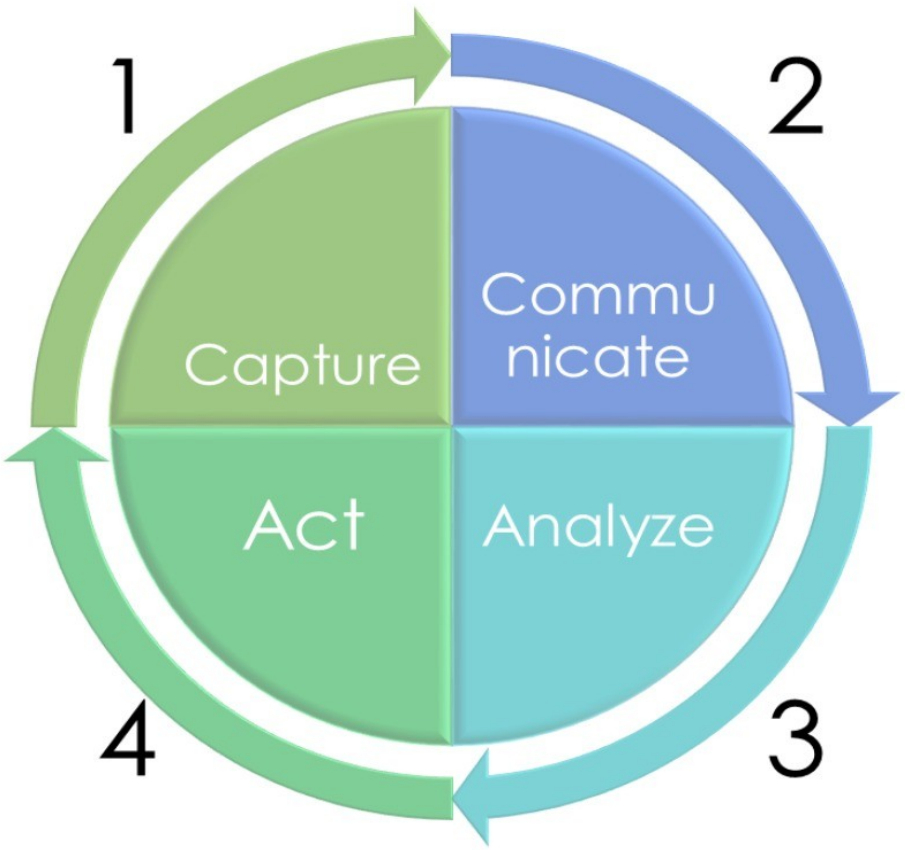
3. ARHITEKTURE IOT PLATFORMI

3.1. Životni ciklus IoT sistema

Ako gledamo strukturu IoT sistema, možemo da primetimo četiri stanja u kojima sistem može da se nađe. Ta četiri stanja su

1. *Capture* (srp. detektuj)
2. *Communicate* (srp. komuniciraj)
3. *Analyze* (srp. analiziraj)
4. *Act* (srp. odradi aktuaciju) [5]

Ova četiri stanja predstavljaju ujedno i životni ciklus svakog IoT sistema. Na slici 5 je dat prikaz životnog ciklusa IoT sistema.



Slika 5: Životni ciklus IoT sistema

Capture

Prvo stanje ciklusa je *Capture* stanje. U ovom stanju se nalaze oni uređaji koji imaju senzore u sebi. Oni mogu da detektuju spoljašnje pojave poput temperature, ambijentalnog

osvetljenja, vlažnosti vazduha, količinu čestica u prostoru. Ove pojave senzori konvertuju u digitalne signale, koji se zatim prosleđuju dalje kroz sistem. [5]

Communicate

Sledeće stanje u ciklusu je *Communicate*. Za ovu fazu postoji preduslov, da postoji veza između uređaja putem neke mrežne tehnologije. U ovoj fazi se ono što je izmereno u senzorima komunicira ka drugim uređajima. Ovo može biti ili ka nekom centralizovanom serveru ili ka nekom konkretnom uređaju. Komunikacija između uređaja u IoT sistemu se vrši preko već postojećih protokola za komunikaciju npr. HTTP, MQTT itd. [5]

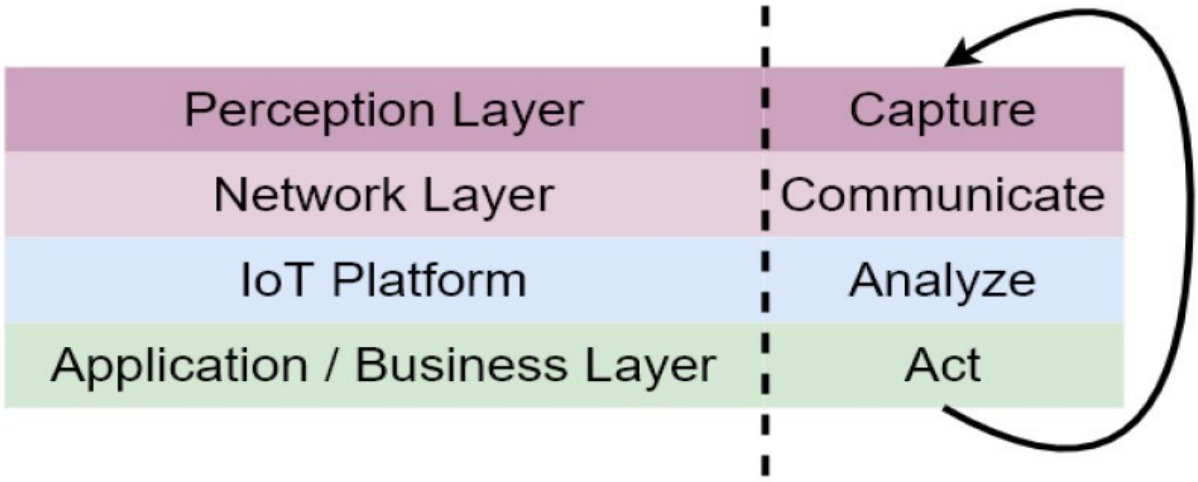
Analysis

Treća faza ciklusa, *Analysis*, je odgovorna za analiziranje podataka koji dolaze sa IoT uređaja. Analize se mogu vršiti na dva načina, u *batch*-u ili u *realnom vremenu*. Batch analiza akumulira pristigle podatke neko vreme, i nakon isteka tog vremena, analizira podatke. Analiza podataka u realnom vremenu podrazumeva da se podaci analiziraju kako pristižu. [5]

Act

Act je finalna faza u životnom ciklusu IoT sistema. U ovoj se fazi, na osnovu rezultata dobijenih u prethodnoj fazi, vrši određena akcija. Akcije koje se izvršavaju se dele na dva tipa. Prvi tip akcije je *vizualizacija podataka* za krajnjeg korisnika. Drugi tip akcije se često naziva i *aktuacija*. Aktuacija predstavlja delovanje nekog uređaja na okolinu u kojoj se nalazi. Npr u sistemu za navodnjavanje, može se instalirati uređaj koji može da pokreće ventil za navodnjavanje. [5]

Na slici 6 je dat prikaz kako se faze u životnom ciklusu IoT sistema podudaraju sa slojevitom arhitekturom IoT sistema, datoj u poglavlju 2.3. Sa leve strane grafika su slojevi u tradicionalnoj slojevitoj arhitekturi IoT sistema, a s desne strane su faze u životnom ciklusu IoT sistema.



Slika 6: Veza životnog ciklusa i arhitekture IoT sistema[5]

Perception (srp. Percepcija, očitavanje) sloj je sinonim za Sensor layer u slojevitoj arhitekturi. Njegov zadatak je u *Capture* fazi, tj da očita podatke sa svakog IoT uređaja (njegovo stanje, temperaturu, vlažnost vazduha, itd). [5]

Network (srp. mrežna) sloj šalje podatke sa IoT uređaja ka višim slojevima. Može koristiti 2G, 3G, 4G, ili odskora i 5G tehnoliju, kao i Wifi ili Bluetooth za prenos podataka. Odgovara *Communication* fazi životnog ciklusa. [5]

Application/Business sloj odgovara poslednjem, Aplikativnom sloju u slojevitoj arhitekturi. Na ovom sloju se upravlja obrađenim podacima koji dolaze iz prethodnog sloja. Ovde se primenjuju određena biznis pravila na neki scenario, npr „smart“ home, „smart“ health itd. Na ovom sloju se vizualizuju podaci, ili se vrši određena aktuacija. Zato ovaj sloj odgovara *Act* fazi u životnom ciklusu.[5]

IoT platform sloj (srp. *sloj IoT platforme*) analizira i perzistira podatke koji dolaze sa IoT uređaja i kreira vizualizacije tih podataka. Sloj IoT platforme odgovara *Analyze* fazi u životnom ciklusu.[5] O ovom sloju će biti posebno reči u narednom delu, naročito o njegovoj arhitekturi, funkcionalnostima i karakteristikama. [5]

3.2. IoT platforma

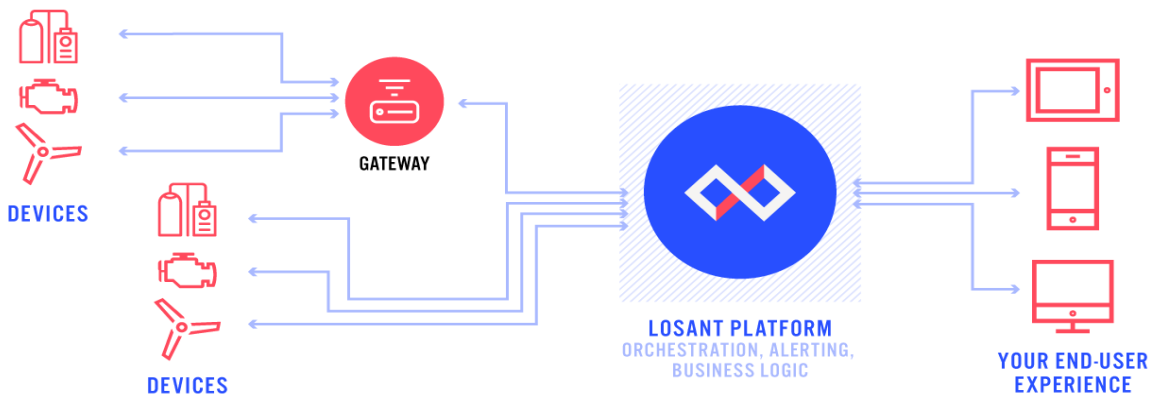
Ideja koja stoji iza IoT platforme je ideja objedinjavanja funkcionalnosti. Kada sve informacije dođu do Cloud sloja, potrebno je sačuvati pristigle podatke, potencijalno izvršiti agregaciju podataka, analizirati ih, pokrenuti aktuacije itd. Kako bi se ovo postiglo predložen je koncept IoT platforme. IoT platforma kombinuje različite softverske alate kako bi se napravila najoptimalnije rešenje za prikupljanje, čuvanje, analizu i upravljanje pristiglim podacima. [6]

Zbog više zadataka koje IoT platforme imaju potrebno je da se izvršavaju na mašinama koje imaju veliku procesnu moć. Zbog velike količine resursa koji se nalaze na Cloud sloju, najčešće se IoT platforme postavljaju tamo. Neke od prednosti koje Cloud sloj nudi su:

- **Neograničena skalabilnost** – Ključna stvar u IoT sistemima, gde protok može biti I po nekoliko terabajta podataka u sekundi. Takođe sistem može neograničeno da raste, odnosno broj novih uređaja koji može da se poveže na sistem je praktično neograničen.

- **Cloud omogućava remote razvoj aplikacija** – Kod IoT sistema uređaji mogu biti razbacani na različitim geografskim lokacijama. Ako su svi imaju pristup Cloud platformi, lakše je menjanje verzije softvera koji se izvršava, popravljanje bugova itd. [6]

IoT platforme imaju različite arhitekture, ali imaju slične osnovne funkcionalnosti I sličan položaj u IoT sistemu. Na slici 8 je dat prikaz topološke pozicije IoT platforme u IoT sistemu.



Slika 7: Topološki položaj IoT platforme u IoT sistemu[8]

IoT platforma se najčešće nalazi između fizičkih senzora I krajnjih korisnika. IoT platforma komunicira sa senzorima i čita podatke koje ti senzori šalju. Nakon toga, analizira podatke, vizualizuje ih krajnjem korisniku, i ako je potrebno vrši se određena aktuacija.

Funkcionalnosti koje IoT platforma mora da nudi su date na slici 9. Iako ne postoji standardizovano rešenje za IoT platforme, u svim arhitekturama se mogu uočiti ove četiri funkcionalnosti. Podrazumevana osnovna funkcionalnost je mogućnost povezivanja IoT platforme sa ostalim uređajima.

Slika 8: Funkcionalnosti IoT platforme

Perzistencija podrazumeva da IoT platforma može da čuva primljene podatke. Uz samo čuvanje podataka, može se vršiti I agregaciju podataka. Tako se može olakšati vizualizacija ili analiza podataka. IoT platforma treba da ima podršku za izvršavanje upita. IoT platforma treba da podršku za izvršenje CRUD operacija nad podacima.

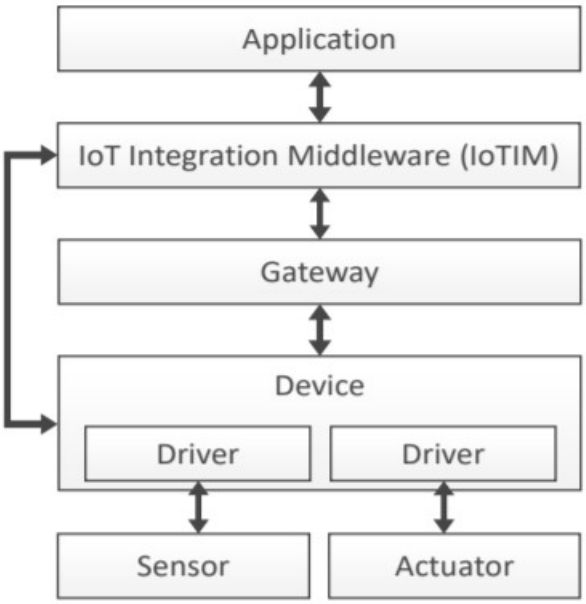
Još jedna od funkcionalnosti koje IoT platforma treba da podržava je I analiza podataka. Analizom podataka bi trebalo utvrditi trendove koji se javljaju u podacima, odrediti klastere među podacima itd. Na ovaj način, sem što se ima trenutni uvid u stanje sistema, može se na osnovu trenda predvideti buduće stanje sistema, I pravovremeno reagovati. Na kraju, sistem bi trebalo da ima mogućnost aktuacije uređaja I vizualizacije podataka.

Na slici 9 je dat dijagram komponenti generičke IoT aplikacije koja uključuje I IoT platformu. Najniži sloj je *device* sloj (srp. Sloj uređaja). Na ovom sloju se nalaze senzori I drajveri za senzore. Ovaj sloj služi da detektuje promene u okruženju senzora. Pomoću drajvera I pomoćnih mikroračunara se izmerene vrednosti ili. *gateway* sloju (srp. gateway – kapija) ili direktno IoT platformi. Ako uređaj ne može sam da komunicira sa drugim uređajima, povezan je na gateway. Gateway pruža mehanizme za prevođenje između različitih protokola, formata poruka itd. IoT middleware sloj predstavlja samu IoT platformu. Ona služi da čita podatke koji dolaze sa senzora, da ih obrađuje I vrši aktuaciju uređaja. Sve što IoT platforma detektuje može da koristi aplikativni sloj. Na ovom sloju se nalazi aplikacija koja za vršenje svojih biznis funkcionalnosti koristi IoT platformu. [7]

Neki od scenarija gde je IoT platforma korisna:

- **Monitoring sistemi niske snage** – Kod sistema za parking npr. postojao bi senzor koji bi pratio broj slobodnih mesta I sporadično slao obaveštenja tek kad se broj mesta promeni. [9]

- **Poljoprivredni sistemi** – Kod poljoprivrednih sistema, mogu se stalno čitati vrednosti o vlažnosti vazduha, temperaturi, intenzitetu svetlosti, vlažnosti zemlje. Na osnovu ovih podataka mogu se donositi odluke o sadnji novih biljaka ili navodnjavanju postojećeg zemljišta. Takođe se sistem može lako proširiti, ako se postave novi senzori. Dovoljno je samo podići još jednu instancu IoT platforme.



Slika 9: Dijagram komponenti generičke IoT aplikacije[7]

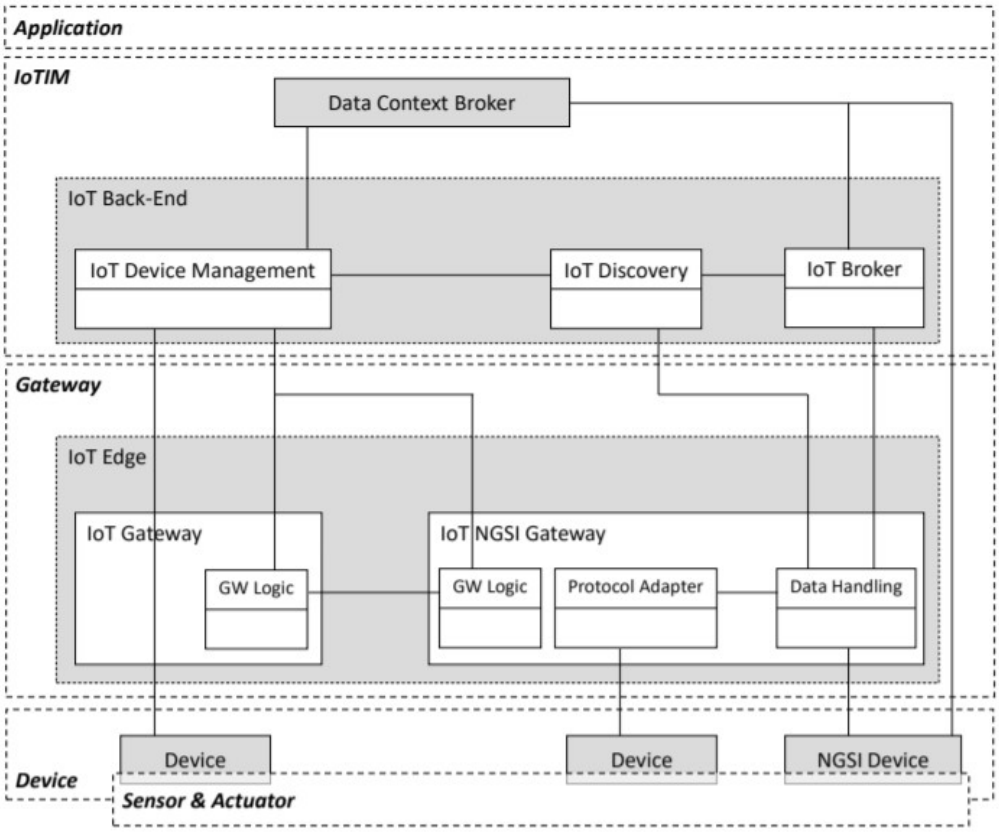
3.3. Pregled postojećih implementacija

U poslednjih par godina razvijeno je vise stotina arhitektura IoT sistema koji koriste IoT platforme. Neki od primera su AWS IoT, FIWARE, OpenMTC i SmartThings. Platforme se međusobno jako razlikuju. Iako postoje tendencije da se objedine oko jednog standarda, trenutno to ne deluje moguće. Pošto ne postoji jedinstveni standard, vrlo često se terminologija razlikuje od tima do tima. Međutim sva rešenja pružaju slične funkcionalnosti, omogućavaju različitim uređajima da pristupe platformi i da se njihovi podaci obrade, i da se upravljanje automatizuje kroz prikupljene podatke. [7]

U narednoj sekciji biće obrađena 2 IoT sistema koji koriste IoT platformu, FIWARE i OpenMTC. Njihova arhitektura biće upoređena sa generičkom arhitekturom IoT sistema.

FIWARE

Na slici 10 je data arhitektura FIWARE IoT sistema. Takođe na slici je prikazano kako se koji deo arhitekture mapira na generički IoT sistem.[7]



Slika 10: Arhitektura FIWARE IoT platforme[7]

FIWARE je projekat Evropske Unije i Evropske komisije. Platforma je *cloud-based,* što znači da se većina njenih komponenti nalazi na cloud-u. Device sloju iz generičkog modela odgovara device sloj u FIWARE sistemu, uz jednu razliku. Ta razlika je da ovaj sistem pravi razliku između NGSI uređaja i uređaja koji ne podržavaju ovaj protokol. [7] NGSI-LD je informacioni model i API koji služi za razmenu kontekstualnih podataka. Definisao ga je

Evropski Institut za Standardizaciju Telekomunikacija. Široku primenu je našao u IoT tehnologiji, tj kod pametnih gradova, pametnih bolnica itd. [10]

Tako u ovom sistemu postoje i dva tipa gateway-a, gateway za NGSI uređaje i gateway za non-NGSI uređaje. Sistem podržava i senzorski i aktuatorski tip uređaja. Uređaji mogu da komuniciraju direktno sa IoT platformom ili preko gateway-a koji su za njih predviđeni. Gateway-i se najčešće nalaze na Edge sloju sistema. [7][4]

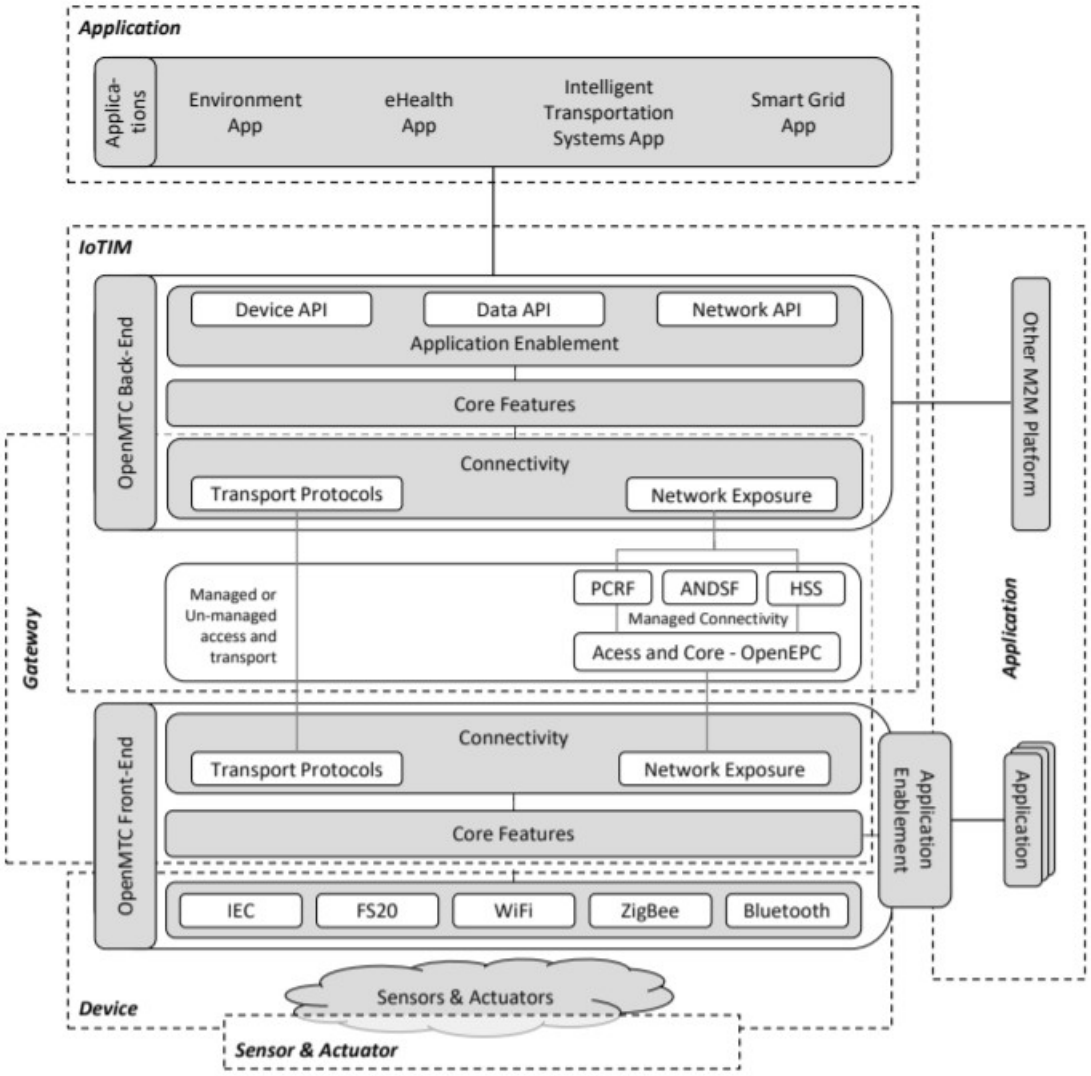
IoT platforma se u ovom sistemu naziva IoT Back-End. Tri glavne komponente IoT platforme u ovoj arhitekturi su: [7]

- **IoT Device Management**
- **IoT Discovery**
- **IoT Broker**

IoT Device Management sloj služi za upravljanje uređajima u sistemu. IoT Discovery komponenta služi za detekciju uređaja u sistemu. IoT Broker služi kao broker poruka u sistemu. [7]

OpenMTC

Na slici 11 data je arhitektura OpenMTC sistema.



Slika 11: Arhitektura OpenMTC sistema[7]

OpenMTC je open-source cloud platforma. Na slici 11 je takođe dato i poređenje komponenti ovog sistema sa generičkim sistemom. OpenMTC se sastoji iz sledećih komponenti: [7]

- **Front-end komponente**
- **Back-End komponente**
- **Senzora i aktuatora**
- **Aplikativnih komponenti**
- **Connectivity komponenti**

Kod OpenMTC sistema senzori i aktuatori kao i komunikacioni protokoli koji služe da ostvare komunikaciju između Front-enda i senzora i aktuatora, odgovaraju Device sloju kod generičkog IoT sistema. Core Feature-i sistema i Connectivity komponenta služe kako bi prevodili sadržaje poruka iz jednog formata u drugi, i kako bi se poruke slale odgovarajućim protokolom između Front-enda i senzora odnosno aktuatora. Zbog toga ove dve komponente odgovaraju Gateway sloju kod generičkog modela. [7]

Gateway sloju takođe odgovara i Connectivity komponenta Back-End sloja. Ona služi da upakuje sadržaj koji ide od Back-End komponente ka Front-end komponenti u odgovarajući format i da iskoristi odgovarajući transportni protokol. Samu srž sistema čine Core Feature-i Back-End sloja, deo Connectivity komponente , kao i Application Enablement komponenta i zajedno čine IoT platformu. Application Enablement sloj služi da olakša povezivanje aplikacija koje koriste funkcionalnosti platforme. [7]

Aplikacije se takođe mogu direktno vezati na Back-End komponentu ili na Front-end komponentu, bez povezivanja na Application Enablement sloj. Gde se koja aplikacija povezuje zavisi od toga koji stepen obrade nad podacima želi i koliko je u stanju da nezavisno komunicira sa sistemom. [7]

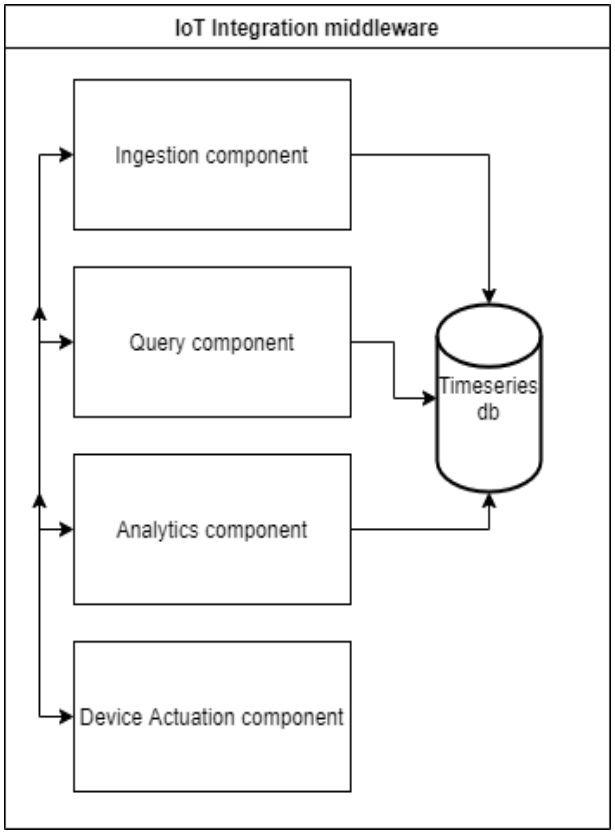
U narednom delu biće reči o dve najpopularnije arhitekture IoT platformi. Monolitnoj arhitekturi i mikroservisnoj arhitekturi. Razmatraće se način implementacije funkcionalnosti IoT platformi kroz ove dve arhitekture. Na kraju će biti dat uporedni pregled ove dve arhitekture, kao i prednosti i mane svake od njih.

3.4. Monolitna arhitektura

Monolitna arhitektura je do pre par godina bila standard u projektovanju web aplikacija. To je intuitivan i praktičan način izrade aplikacija. Ideja je da se sve funkcionalnosti implementiraju u jednom izvršnom fajlu. Taj fajl se zatim postavlja na server. Za kod koji se izvršava postoji jedinstven repozitorijum i svi timovi imaju pristup tom repozitorijumu. Funkcionalnosti koje nudi aplikacija se struktuiraju kao skupovi servisa i kontrolera koji se međusobno pozivaju. Na ovaj način se kreira **jako spregnut** sistem. Da bi se kod mogao kompajlirati, sve njegove komponente (svi servisi i kontroleri) moraju da budu funkcionalne. Kako bi sistem mogao da funkcioniše korektno sve njegove komponente moraju da funkcionišu korektno. Kako bi se što ranije detektovale eventualne greške u sistemu, kako bi se detektovale regresije koje nove promene mogu da uvedu, smišljeni su unit i integration testovi. Unit testovi služe da testiraju funkcionalnosti individualnih komponenti, a integration testovi služe da testiraju međusobnu interakciju komponenti u sistemu. Takođe jedan od izazova pri razvoju aplikacija u monolitnoj arhitekturi je ažuriranje pojedinih delova sistema. Pošto je sistem jako spregnut, često se dešava da, pri ažuriranju jedne do komponenti sistema, moramo da menjamo i sva mesta gde tu komponentu (servis/kontroler) koriste ostali servisi ili kontroleri, što može da bude jako problematično i da uvede regresije i eventualne nekonzistentnosti u sistemu. [11][12]

IoT platforme treba da imaju funkcionalnost da sačuvaju pristigle podatke, da izvršavaju upite nad podacima, rade analize podataka i izvršavaju aktuaciju uređaja. U poglavlju 3.2. dat je predlog osnovne arhitekture IoT sistema na slici 9. Na slici 9 se uočava IoT Integration

middleware (drugo ime za *IoT platformu*) sloj koji je srž sistema i tu su objedinjene sve navedene funkcionalnosti platforme. Pošto se on izvršava na cloud sloju, najčešće arhitektura tog sloja odgovara arhitekturi web aplikacije. Na slici 12 je dat strukturni dijagram monolitnog IoT Integration middleware sloja.



Slika 12: Strukturni dijagram monolitnog IoT Integration middleware sloja

Sve funkcionalnosti su grupisane u komponente. Sve komponente mogu da međusobno da komuniciraju. Sve se nalaze u istoj aplikaciji. Baza podataka je ista za svaku od komponenti, tj za celu aplikaciju. Ako bilo koja komponenta sistema nije funkcionalna, ceo sistem je oboren.

Prednosti monolitne arhitekture su:

- *jednostavna za razvoj* – sav kod se nalazi u jedinstvenom repozitorijumu. Sve komponente mogu koriste funkcionalnosti svih drugih komponenti. Nema nikakve latencije u komunikaciji između komponenti.
- *jednostavna za postavljanje na server* – na server se postavlja jedinstveni izvršni fajl (WAR, .exe itd).

- *jednostavno skaliranje* – aplikacija se jednostavno može skalirati tako što se više instanci aplikacije izvršava. Ispred tih instanci je potrebno da se nalazi load balancer koji bi adekvatno balansirano opterećenje između aktivnih instanci. [12]

U trenutku kada kod u aplikaciji poraste i kada se tim koji na radi na aplikaciji poveća, mane monolitnog sistema postaju primetne i usporavaju razvoj aplikacije i ometaju dodavanje novih funkcionalnosti. Neke od mana monolitne arhitekture su: [12]

- *teže upoznavanje novih članova tima sa projektom* – s porastom količine koda i funkcionalnosti, postaje sve teže da se novi članovi tima upute u projekat. Aplikacija postaje sve teža za razumevanje i modifikaciju. Ovo može jako da uspori razvoj aplikacije.

- *teže održavanje modularnosti koda* – Pošto između komponenti/modula nema čvrstih granica, modularnost koda opada s vremenom. Ovo često rezultuje u opadanju kvaliteta koda.

- *duže vreme startovanja aplikacije* – velikim aplikacijama treba više vremena da se pokrenu. Glomaznim monolitnim aplikacijama treba i po nekoliko sati da se startuju.

- *continuous deployment je otežan* – kako bi se ažurirala jedna komponenta aplikacije potrebno je da se cela aplikacija ponovo postavi na server. Kreiranje builda i deployment može da traje i po nekoliko sati, što može biti problematično ako je neka promena hitno potrebna. Takođe pozadinski job-ovi koji se izvršavaju mogu biti prekinuti i time podaci u sistemu mogu biti ostavljeni u nekonzistentnom stanju što može izazvati potencijalne probleme. Postoji i šansa da komponente koje nisu menjane ne startuju kako treba zbog promene nekog interfejsa ili servisa od kog su zavisne. Sve ovo obeshrabruje tim da stalno menja verzije aplikacije.

- *skaliranje može biti problematično* – aplikacije projektovane u monolitnoj arhitekturi mogu da se skaliraju podizanjem više instanci iste aplikacije. Ovo može da bude problematično i može zahtevati mnogo resursa. Vrlo često je i neefikasno, jer je obično u jednom trenutku samo jedan deo sistema opterećen prilivom podataka. Kako bi se odgovorilo na ovaj priliv podataka umesto da se skalira samo deo koji je pod opterećenjem, skaliraće se cela aplikacija.

- *jednom kad se izabere stek tehnologija nema promene* – tehnologije se biraju na početku razvoja aplikacije. Te tehnologije mogu da zastare ili da prestane podrška za njih. Ako se želi prelaz na neku savremenu tehnologiju potrebno je da se cela aplikacija prepiše, što može biti veliki posao.[12]

3.5. Mikroservisna arhitektura

S porastom kompleksnosti web aplikacija, mikroservisna arhitektura je postala standardna arhitektura aplikacija. Ideja kod mikroservisne aplikacije je da se ona struktuira

kao skup nezavisnih servisa. Ti servisi su *slabo spregnuti*, što znači da ako se jedan od servisa ne funkcioniše, ostatak sistema neće biti ugrožen. Ovim se postiže visok up-time sistema. Servisi mogu da sarađuju međusobno i to je sprovedeno na takav način da se nezavisnost servisa ne remeti. [13]

Svaki servis je:

- Lak za održavanje i testiranje
- Brzo se postavlja na server
- Bavi se samo jednom funkcionalnošću sistema
- Lako skalabilan
- Slabo je spregnut sa ostalim servisima[13]

Komunikacija između servisa se odvija na dva načina:

- *Sinhrono* – putem sinhronog protokola poput HTTP ili gRPC.
- *Asinhrono* – korišćenjem brokera poruka [13]

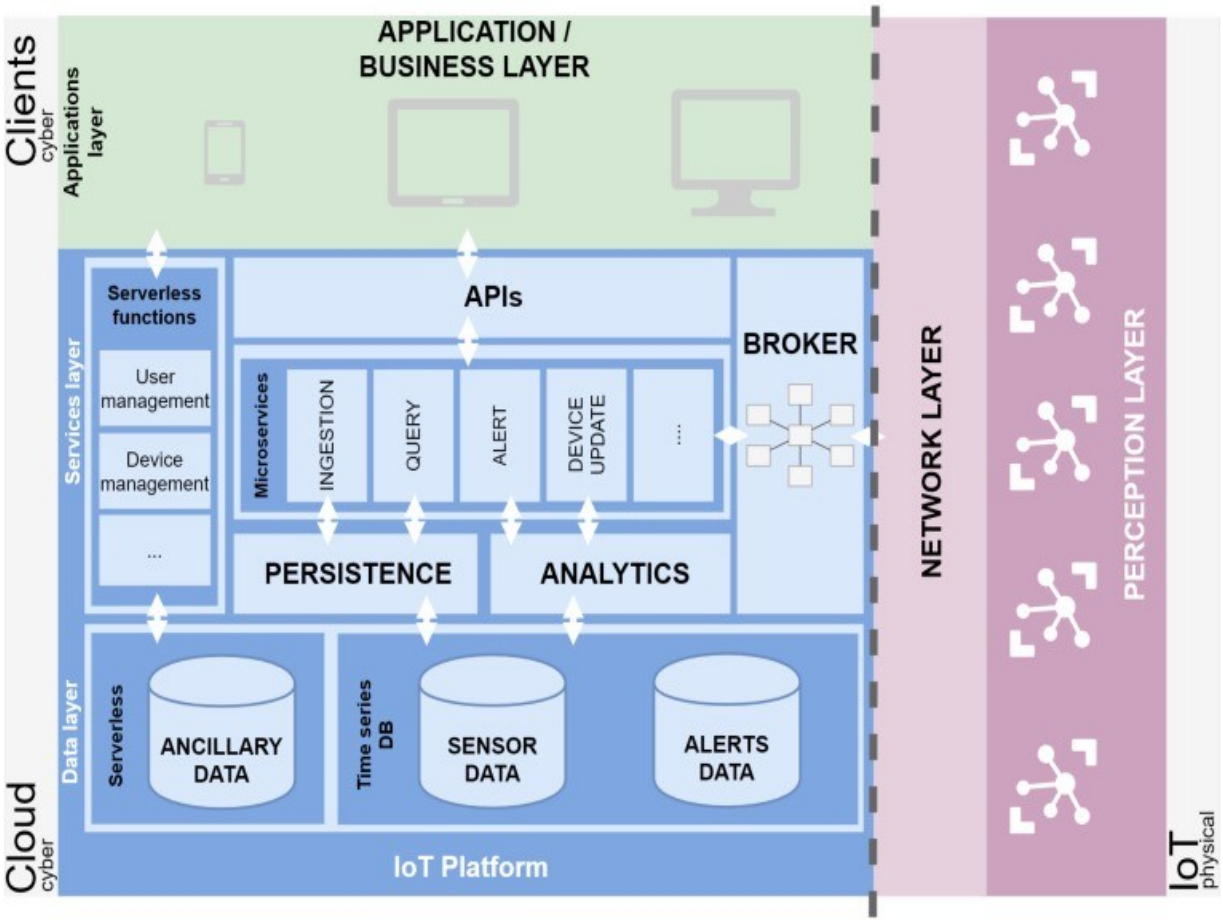
Najčešće svaki od servisa ima svoju bazu podataka, kako bi se smanjila zavisnost komponenti u sistemu. Za očuvanje konzistentnosti podataka u sistemu koristi se SAGA obrazac. SAGA obrazac omogućava da se izvršavaju transakcije koje zahvataju više servisa. Omogućavaju da sistem bude u eventualno konzistentnom stanju. Velika prednost korišćenja SAGA u odnosu na distribuirane transakcije je ta što ne moraju svi servisi da budu dostupni u trenutku dešavanja transakcije. [13]

Pošto zahtev koji dolazi na sistem može da se prostire na više servisa, korisno je pratiti gde se kretao. Za ovu svrhu osmišljeno je distribuirano trasiranje koje prati zahtev koji je došao u sistem i svu interservisnu komunikaciju koja se dešava da se pribavi rezultat. Health checking je takođe važan deo mikroservisne arhitekture. To znači da servisi periodično signaliziraju platformi na kojoj su deploy-ovani da je sve u redu s njima i da su prisutni. Na ovaj način se dobija predstava u realnom vremenu o aktivnosti servisa. [13]

Kako bi se izbeglo da korisnici aplikacije moraju da kontaktiraju više servisa i da znaju adrese svakog od njih, osmišljena je *API gateway* komponenta. Ova komponenta predstavlja jedinstvenu tačku ulaska u sistem i predstavlja viši nivo apstrakcije. Korisnici znaju adresu API gateway komponente i njemu šalju zahteve. API gateway zatim prosleđuje zahteve ka odgovarajućim servisima i prikuplja rezultate. API gateway komponenta često radi zajedno sa Service Discovery komponentom. Kada se servis podigne, on šalje heartbeat poruku Service Discovery komponenti, signalizirajući da je tu i da je instanca zdrava. Ove poruke se periodično razmenjuju. Service Discovery komponenta vrši i balansiranje opterećenja. To znači da pristigle zahteve balansirano šalje aktivnim instancama servisa. Na ovaj način se smanjuje opterećenje pojedinačnih servisa i omogućava se brža obrada korisničkih zahteva. [13]

IoT platforma projektovana u arhitekturi mikroservisa

Mikroservisna arhitektura se može iskoristiti pri projektovanju IoT platforme. Na slici 13 je dat primer mikroservisne arhitekture IoT platforme. Na slici 13 su takođe označeni i sva 4 dela arhitekture IoT sistema.



Slika 13: Primer IoT sistema gde je IoT platforma ima mikroservisnu arhitekturu[5]

Neki od razloga za korišćenje mikroservisne arhitekture pri projektovanju su:

- Nezavisni deployment servisa
- Nezavisno skaliranje servisa
- Jasna podela odgovornosti kod servisa
- Dostupnost cele aplikacije se povećava, jer pad jednog servisa neće uticati na ostale servise[5]

Servisi koji su izdvojeni na predlogu arhitekture na slici 13 su:

- *Ingestion servis* (srp. s*ervis za prihvatanje podataka*) – služi da prihvati podatke koji dolaze sa senzora i da ih perzistira u nekoj time series bazi podataka. Ovaj servis takođe može da izvršava job-ove u pozadini koji agregiraju podatke koji su pristigli u određenom vremenskom intervalu.

- *Query servis* (srp. *servis za upite*) - ima mogućnost izvršavanja upita nad podacima koji su upisani u time series bazu podataka.

- *Alert/Analytics servis* (srp. *servis za analizu podataka*) – ovaj servis analizira prikupljene podatke, i detektuje da li su se javile ili će se javiti neke anomalije koje mogu da utiču na spoljni sistem

- *Device Update/Actuation servis* (srp. *servis za aktuaciju uređaja*) – služi da šalje ažuriranja uređajima koji su deo sistema, kao i eventualne signale za aktuaciju na osnovu toga što je zaključeno u Analytics servisu. [5]

Treba primetiti da su servisi izdvajani u skladu sa funkcionalnostima IoT platforme dati na slici 8.

Na slici 13 se vidi da svaki od servisa ima svoju bazu podataka. Time series baze podataka koriste Ingestion i Query servis. Analytics servis može da koristi bilo koju SQL ili NoSQL bazu podataka. Servisi međusobno komuniciraju putem brokera poruka. Svi senzori šalju podatke na broker poruka putem MQTT protokola. Te podatke zatim prikuplja Ingestion servis i perzistira ih. Ako korisnik zatraži neki upit, taj zahtev se prosleđuje Query servisu. Sve pristigle podatke analizira Analytics servis, ili odmah po pristizanju ili po batch-evima. Ako je potrebna neka aktuacija uređaja, takav zahtev se prosleđuje ka Device update servisu. On takođe može da izvršava i pozadinske job-ove koji mogu da izvršavaju periodičnu aktuaciju uređaja. [5]

U sistemu se može koristiti i API gateway komponenta, koja pruža jedinstvenu ulaznu tačku u sistem. Preko API gateway-a spoljni korisnici mogu da koriste funkcionalnosti svih servisa.

Prednosti

Neke od prednosti koje nudi arhitektura mikroservisa su:

- *Olakšava deployment aplikacija* – za deployment nove verzije nekog servisa, nema potrebe da se ponovo deploya ceo sistem. Kreira se samo novi build tog servisa i deploy-a se na server

- *Olakšava održavanje servisa* – svaki od servisa je relativno mali i lak za razumevanje i dodavanje novih funkcionalnosti ili prepravku starih

- *Lakše testiranje servisa* – servisi su manji, brži i jednostavniji za testiranje

- Manje vreme startovanje aplikacija pošto se pojedinačni servisi brže pokreću od monolitne aplikacije

- Povećana dostupnost aplikacije, pošto pad nekog od servisa ne utiče na ostale

- Svaki od servisa se može pisati u drugačijoj tehnologiji, time timovi koji rade na aplikaciji mogu biti raznovrsni [5]

Mane

Neke od mana koje donosi mikroservisna arhitektura su:

- Interservisna komunikacija uvodi dodatnu kompleksnost u sistem
- Komplikovanije je implementirati zahteve koji se prostiru kroz više servisa
- Testiranje interakcije između servisa je teže
- Kompleksnije je održati podatke u sistemu u konzistentnom stanju

3.6. Komparativna analiza arhitektura

U tabeli 1 je dat uporedni prikaz monolitne i mikroservisne arhitekture.

Karakteristike	**Monolitna arhitektura**	**Mikroservisna arhitektura**
Skalabilnost	Lako skaliranje cele aplikacije. Nemoguće skaliranje samo 1 komponente sistema.	Lako skaliranje svake komponente sistema i jednako raspoređivanje opterećenja po svim replikama.
Vreme potrebno za inicijalno projektovanje	Relativno brzo se sastavlja inicijalna arhitektura aplikacije	Duže vreme potrebno za projektovanje sistema
Deployment	Kod velikih aplikacija može potrajati i po par sati	Za pojedinačne servise može trajati po nekoliko minuta
Testiranje	Lakše testiranje komponenti i interakcija između njih	Teže testiranje interakcija između komponenti
Povezanost komponenti u sistemu	Čvrsto spregnute komponente	Slabo spregnute komponente
Vreme potrebno za upoznavanje člana tima s projektom	Kod stvarno velikih projekata, upoznavanje može trajati i po godinu dana	Do par meseci
Vezanost za tehnološki stek izabran na početku razvoja aplikacije	Jednom izabran ne menja se. Ako se želi promena steka, aplikacija se mora prepisati.	Svaki servis može biti pisan u različitim tehnologijama

Tabela 1: Uporedna analiza mikroservisne i monolitne arhitekture

4. IMPLEMENTACIJA IOT PLATFORME ZASNOVANE NA ARHITEKTURI MIKROSERVISA

U nastavku je data implementacija IoT platforme. Platforma može biti upotrebljena u bilo koju svrhu, tj. projektovani sistem je opšteg tipa. Neke od svrha za koje se sistem može upotrebiti su: agrikultura, medicina itd.

4.1. Opis sistema

Platforma je zasnovana na mikroservisnoj arhitekturi. Izdvojena su 4 servisa:

- *Ingestion servis*
- *Query servis*
- *Analytics servis*
- *Device servis*

Platforma ima za cilj da poveća skalabilnost i dostupnost sistema. Treba da omogući i osnovne funkcionalnosti IoT sistema. Podaci koji pristižu sa senzora treba da se perzistiraju u sistem. Potrebno je omogućiti REST API pomoću kojih se mogu izvršavati upiti nad podacima. Te podatke treba analizirati, i ako se predvidi da u nekom trenutku u budućnosti može doći do problema, potrebno je pokrenuti aktuaciju. Kada se aktuacija završi potrebno je da uređaj koji je završio aktuaciju obavesti platformu da je aktuacija završena. Ovo je primer *SAGA obrasca* koji je implementiran u sistemu. Korisnik može i sam da zakaže aktuaciju korišćenjem REST endpointa.

SAGA obrazac se koristi u mikroservisnoj arhitekturi za očuvanje konzistentnosti podataka u sistemu. Određena biznis logika u sistemu može zahtevati učešće više servisa. SAGA predstavlja skup lokalnih transakcija (transakcija koje se izvršavaju u svakom od servisa koje biznis logika obuhvata). Redosled lokalnih transakcija je strogo utvrđen. Kada se lokalna transakcija izvrši, taj servis emituje poruku kojom se obaveštava da je ona završena i da sledeća transakcija može da započne sa izvršavanjem. Ako se neka od lokalnih transakcija izvrši neuspešno, svaki od prethodnih servisa, u obrnutom redosledu, izvršava kompenzujuću akciju, da se poništi dejstvo prethodne transakcije. Na ovaj način se čuva konzistentnost podataka u sistemu. [14]

SAGE se dele na *orkestrirane* i *koreografisane*. Kod orkestriranih SAGA postoji poseban servis tzv. o*rkestrator* koji koordiniše izvršavanje lokalnih transakcija. Kod koreografisanih saga, servisi tačno znaju koji servis sledi posle njih, tj. nema orkestratora već servisi direktno komuniciraju. [14]

Za potrebe ovog rada je implementirana i aplikacija koja imitira senzore. Aplikacija čita podatke koji se nalaze u tekstualnom fajlu i zatim ih šalje MQTT protokolom na broker, baš kao da je reč o pravim senzorima. Podaci su veštački generisani za potrebe ovog rada. Ukupno ima 150 instanci u skupu podataka. Atributi svake instance su:

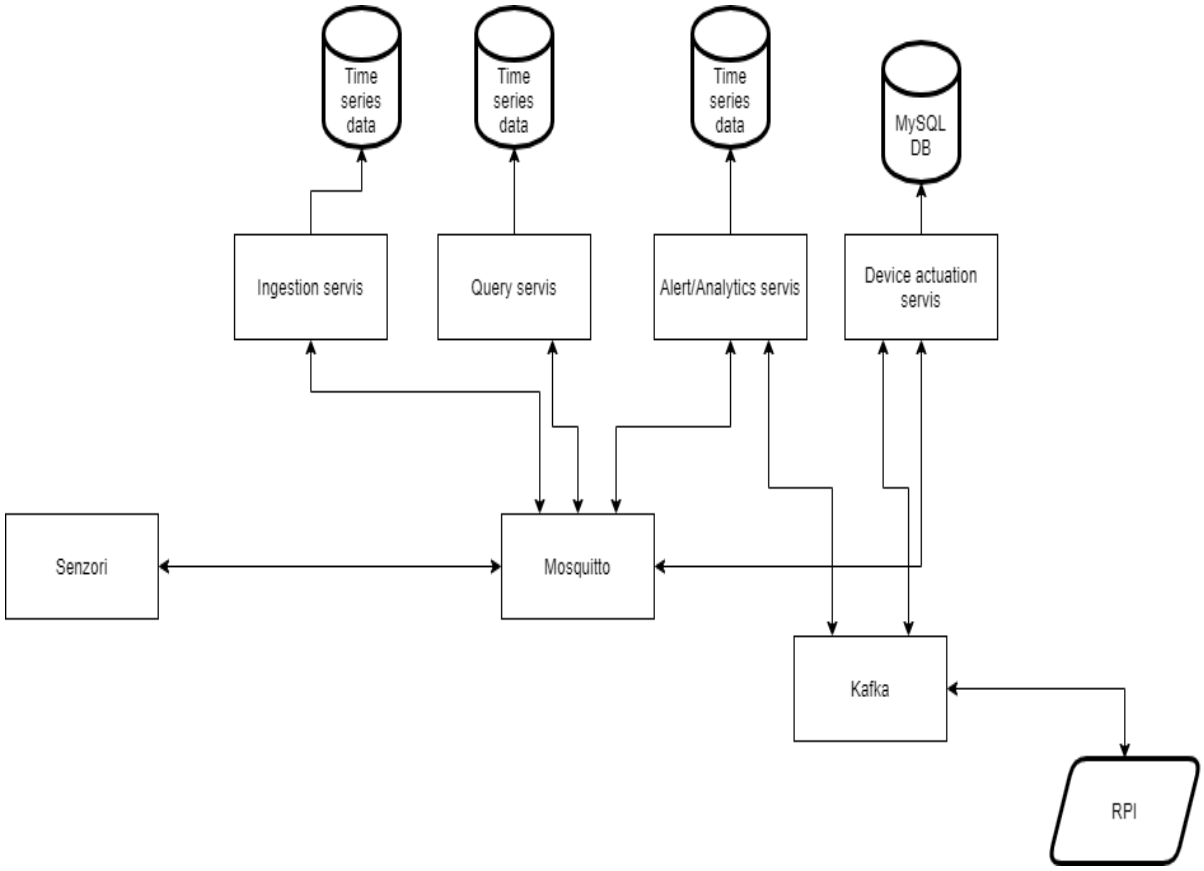
- Intenzitet svetlosti
- Vlažnost zemljišta
- Intenzitet vetra
- Sastav zemljišta

Podaci su podeljeni u tri grupe: *Good*, *Mixed* i *Bad*. Svaka od grupa sadrži po 50 instanci.

Za analizu podataka koristi se višeklasna klasifikacija.

Kako bi se obezbedila i osigurala nezavisnost servisa koriste se brokeri poruka za interservisnu komunikaciju.

Blok dijagram sistema dat je na slici 14.



Slika 14: Blok dijagram IoT platforme

4.2. Korišćene tehnologije

Spring Boot

Spring Boot je jedan od projekata tima koji razvija Spring frejmvork. On olakšava kreiranje i konfiguraciju Spring projekta, ima u sebi ugraĎen server, najčešće Tomcat, tako da je na korisniku da samo kreira svoje aplikacije bez potrebe da brine o konfiguraciji aplikacije. Uključuje osnovne funkcionalnosti Spring MVC projekta, zajedno sa funkcionalnostima Spring Data projekta, pa ima podršku u sebi za rad sa različitim bazama. Uz Spring Boot korišćen je i JPA i Hibernate kao JPA provajder. U ovoj tehnologiji su projektovani individualni servisi. [16]

Apache Kafka

Apache Kafka je open-source distribuirana platforma za strimovanje događaja. Strimovanje događaja je kada događaje koji se dese u sistemu predstavimo kao nekakav tok i taj tok događaja perzistiramo, zbog neke kasnije obrade događaja. Ima 3 karakteristike koje su važne: [15]

- U Apache Kafki se mogu čitati ili upisivati podaci u specifične kanale koji se nazivaju topici, a koji su vezani za jedan tok podataka.
- Tokovi podataka se bezbedno perzistiraju tako da su uvek dostupni.
- Obrada toka podataka u redosledu pojavljivanja. [15]

Kafka funkcioniše tako što *događaje* (koji imaju jedan par key-value informacija, gde je key najčešće informacija o korisniku koji je izazvao promenu, a value je informacija o tome šta se desilo, i jedan timestamp o tome kada se desio događaj) emituje *producer* (srp. proizvođač) na neki topik. Topici imaju ime i najčešće su vezani za jedan tok podataka. Na jedan topik se može pretplatiti više *consumera* (srp. potrošača) koji osluškuju topik i čitaju poruke koje su emitovane. Jedan topik može imati više producera i consumera, i poruke jednog topika se mogu perzistirati neograničeno dugo, dužinu čuvanja poruke programer može definisati u podešavanjima. Jedan topik može biti podeljen na particije, koje omogućavaju paralelizaciju topika i efikasniju podelu poruka među različitim grupama korisnika. [15]

Mosquitto

Eclipse Mosquitto je open-source broker poruka. Podržava MQTT protokol i to verzije 5.0, 3.1.1 i 3.1. Mosquitto je lightweight protokol i njegova upotreba se preporučuje na svim uređajima, od uređaja niske snage do servera. [17]

MQTT protokol pruža lightweight način prenosa poruka korišćenjem publish/subscribe mehanizma. Zbog toga je ovaj protokol pogodan za razmenu poruka u aplikacijama za Internet stvari. Na primer, za razmenu poruka između senzora, računara, mobilnih uređaja, mikrokontrolera itd. [17]

Influx DB

InfluxDB je open-source time series baza podataka. Razvijena je od strane InfluxData kompanije. Pisana je u Go programskom jeziku. Služi za upis i čitanje time series podataka. Upravo to ga čini povoljnim za korišćenje u IoT aplikacijama. [18]

MySQL

MySQL je open-source relacioni database management sistem. Sistem radi kao server, obezbedujući interfejs za pristup relacionim bazama podataka. [19]

Docker

Doker je open-source platforma za kontejnerizaciju. Omogućava programerima da spakuju aplikacije u kontejnere. Kontejneri su standardizovane izvršne komponente koje kombinuju source kod aplikacije sa operatinim sistemom na kome će se izvršavati taj kod. Docker takođe olakšava i deployment i upravljanje kontejnerizovanim aplikacijama. [20]

Java-ML biblioteka

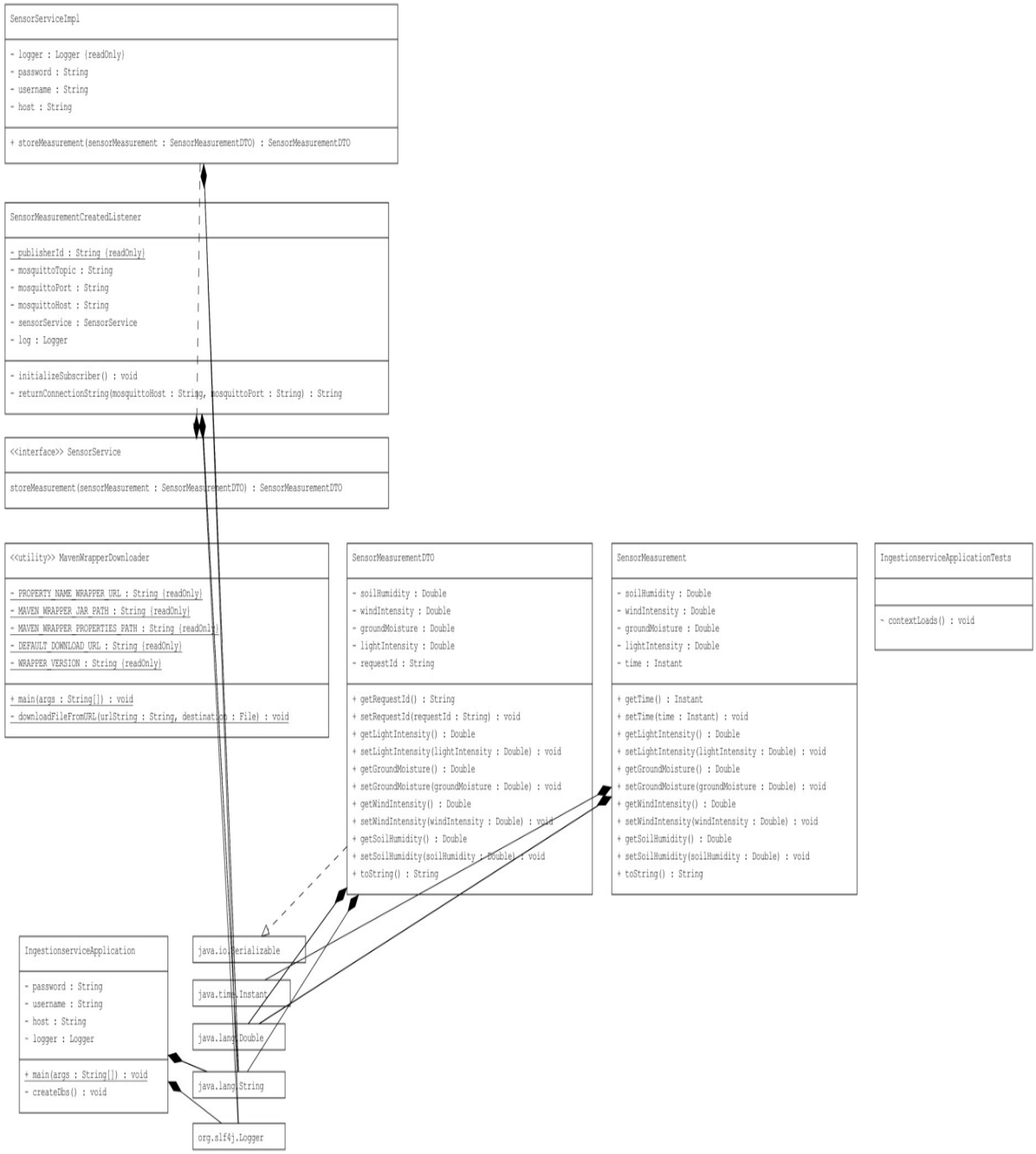
Java-ML biblioteka koja sadrži skup implementiranih algoritama mašinskog učenja. Pisana je za programski jezik Java. Ima zajednički interfejs za svaku vrstu algoritma. Ima podršku za algoritme regresije, klasifikacije, klasterizacije itd. [21]

4.3.Detaljni opis servisa koji čine platformu

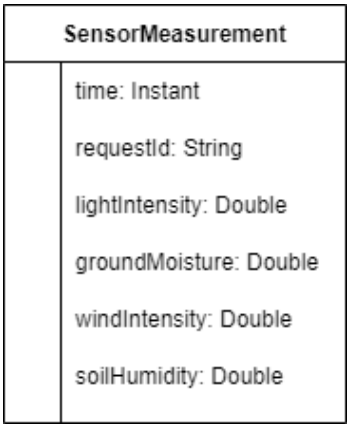
Ingestion servis

Uloga Ingestion servisa u IoT platformi je da perzistira podatke koji su očitani sa senzora. Ingestion servis ima svoju bazu. To je time series baza podataka, InfluxDB. Ingestion servis ima samo jedan listener preko koga sluša poruke koje stižu na Mosquitto broker. Na slici 15 je dat dijagram klasa Ingestion servisa.

Na slici 16 je dat model podataka Ingestion servisa.



Slika 15: Dijagram klasa Ingestion servisa

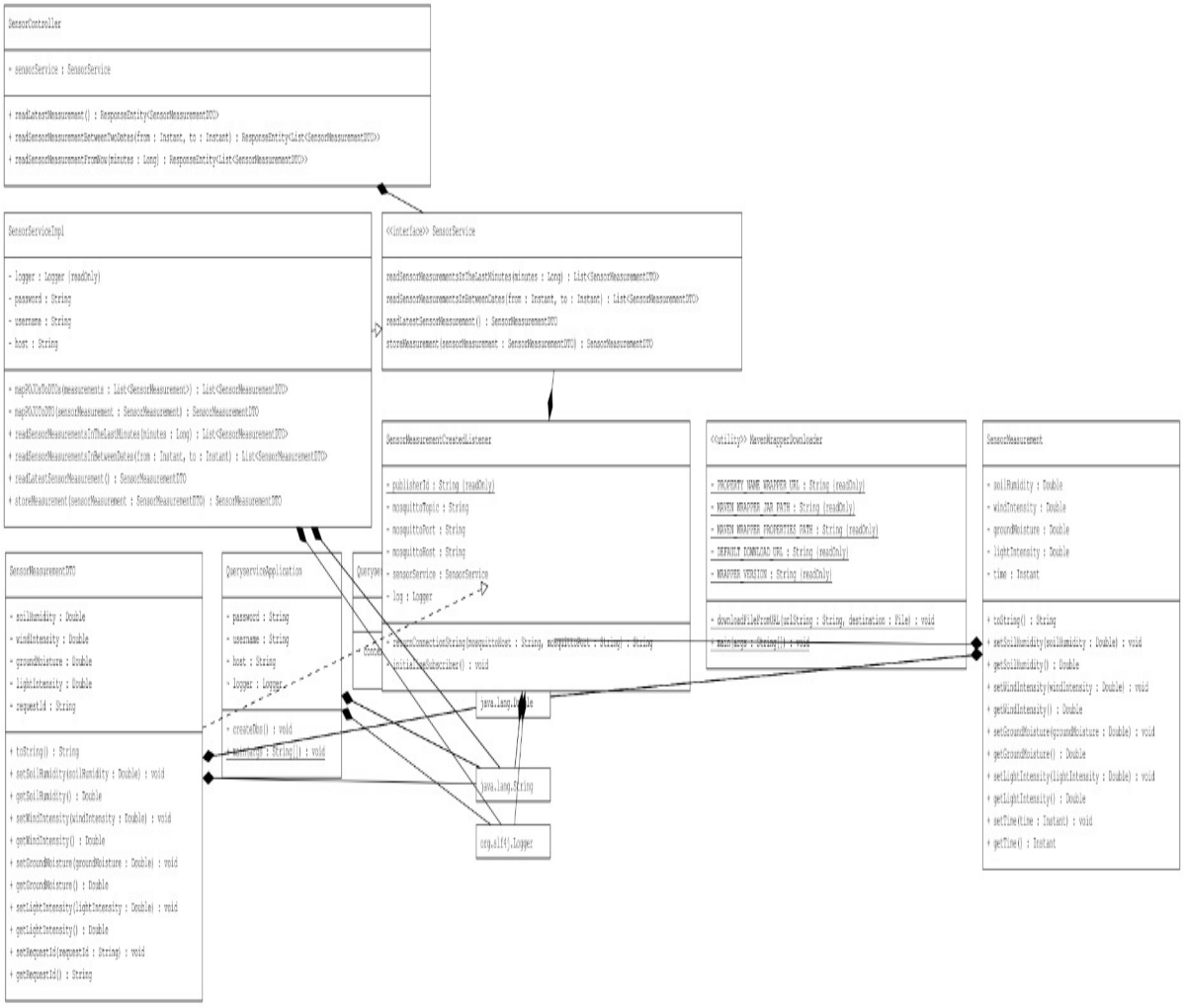


Slika 16: Model podataka Ingestion servisa

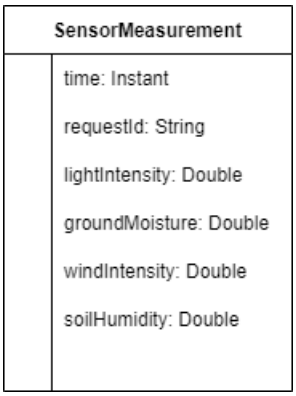
Query servis

Uloga Query servisa je da omogući korisniku da izvršava upite nad perzistiranim podacima. Query servis takođe ima svoju InfluxDB bazu u koju smešta podatke koji pristižu na Mosquitto broker. Takođe ima otvorene REST endpointe preko kojih je moguće izvršavati upite nad bazom. Neki od upita koje je moguće izvršiti su: pročitaj poslednja merenja, pročitaj merenja u poslednjih par minuta, pročitaj merenja između dva datuma itd.

Na slici 17 dat je dijagram klasa Query servisa. Na slici 18 dat je model podataka Query servisa.



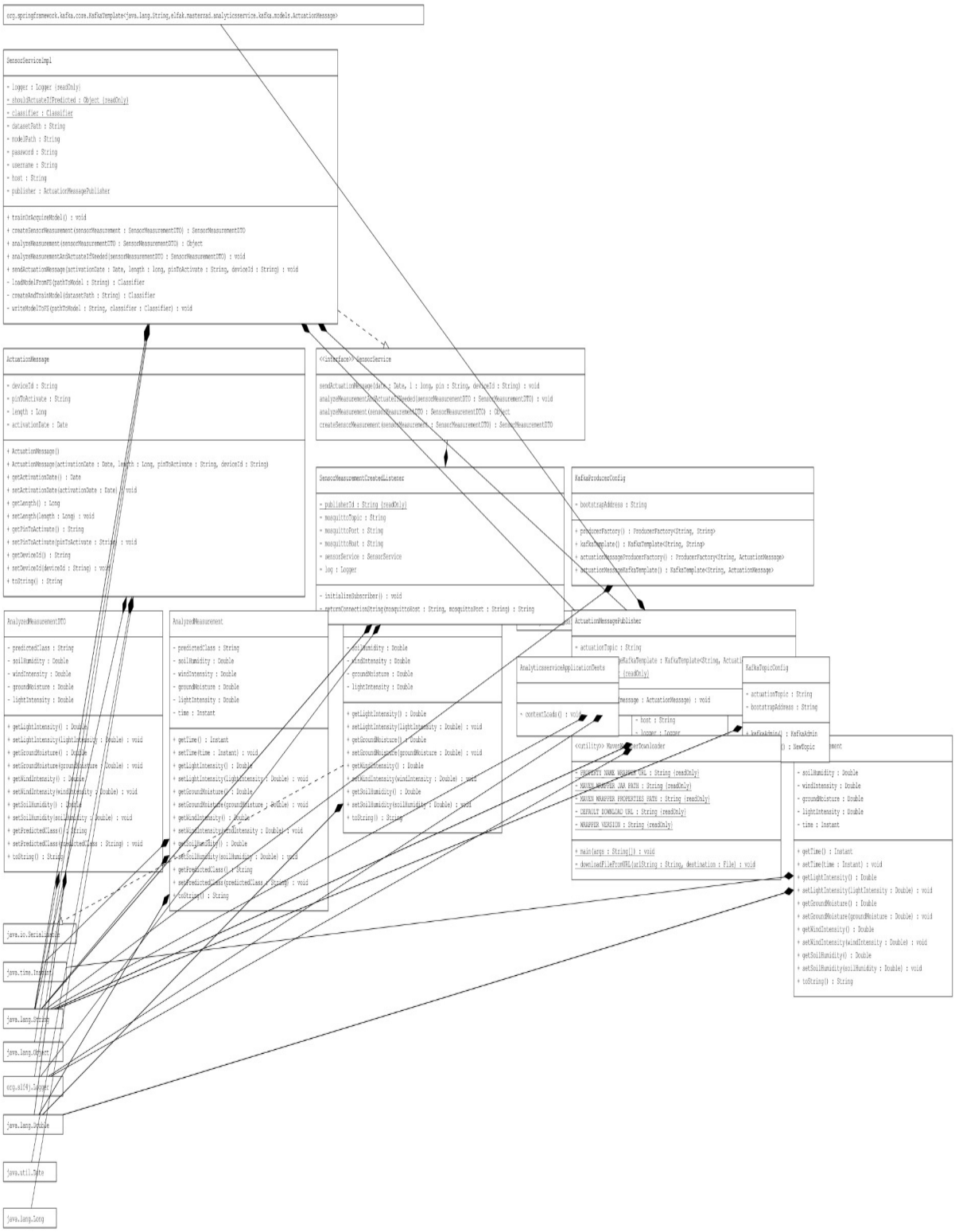
Slika 17: Dijagram klasa Query servisa



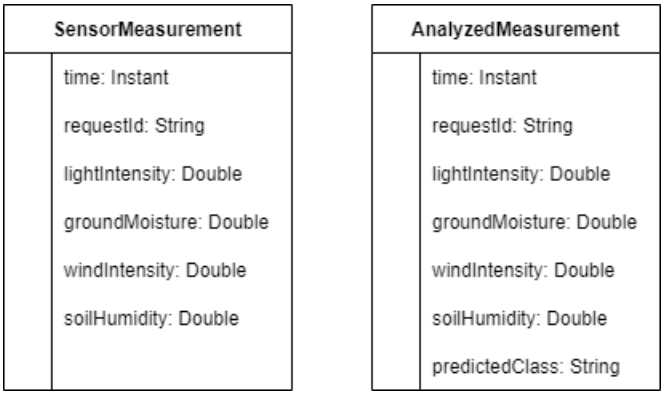
Slika 18: Model podataka Query servisa

Analytics servis

Uloga Analytics servisa je da u pored perzistiranja podataka koji pristižu sa senzora, radi analizu podataka u realnom vremenu. Sve pristigle podatke treba da klasifikuje u tri klase: Good, Bad i Mixed. Ako neki od pristiglih podataka pripada klasi Bad, Analytic servis šalje poruku o aktuaciji na Kafka broker poruka sa informacijama o tome koji uređaj treba da se aktivira, koji pin na tom uređaju, u koje vreme i koliko dugo. Shodno tome, ovaj sistem ne pogoduje real time sistemima. Na slici 19 je dat dijagram klasa Analytics servisa.



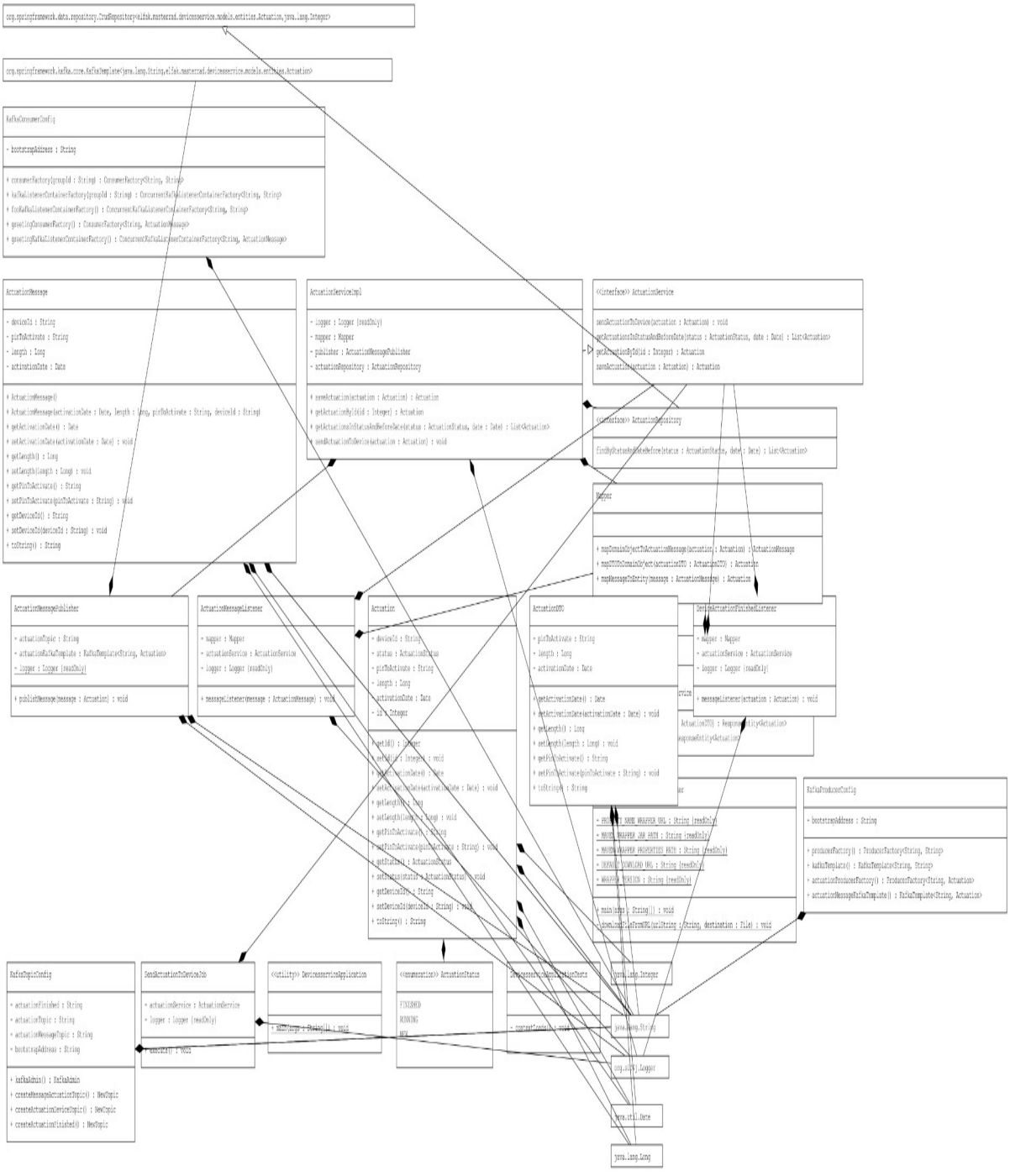
Slika 19: Dijagram klasa Analytics servisa



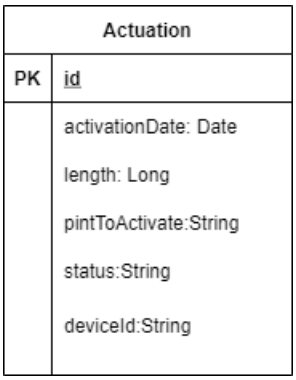
Slika 20: Model podataka Analytics servisa

Device service

Device servis čita poruke o aktuaciji koje mu šalje Analytics servis i čuva ih u svojoj bazi. U pozadini postoji Job koji se izvršava na 5s. Job izvlači sve aktuacije koje su u statusu NEW i čije je vreme izvršenja prošlo. Od tih aktuacija se kreiraju poruke koje se šalju na Kafka broker, na topic koji čitaju device-evi. Sve te aktuacije prelaze u status RUNNING. Uređaj koji prepozna da je poruka za njega, izvršava aktuaciju i šalje informaciju o tome kada je aktuacija završena. Platforma čita tu poruku i tu konkretnu aktuaciju prebacuje u status FINISHED. Device servis ima i REST endpoint kojim korisnik može da zakaže aktuaciju. Device servis koristi MySQL bazu podataka da pamti informacije o aktuacijama.
Na slici 21 dat je dijagram klasa Device servisa.



Slika 21: Dijagram klasa Device servisa



Slika 22: Model podataka Device servisa

4.4. Ključne funkcionalnosti IoT platforme

4.5. IoT platforma – korisnički interfejs

5. PREDLOZI ZA POBOLJŠANJE RAZVIJENOG PROTOTIPA

6. ZAKLJUČAK

Закључак се даје у облику кратких и јасних реченица које представљају осврт на главне доприносе (резултате) у мастер раду. У закључку треба истаћи потенцијалну примену добијених резултата. На крају закључка обично се дају смернице за будући рад у вези теме рада, а имајући у виду намеру да будући читаоци рада добију добру основу за проширивање и унапређење истраживања.

Треба водити рачуна да закључак треба да буде концизан и јасан, без сувишних детаља и понављања претходних реченица. Уобичајено се закључак пише у обиму до 1 странице А4 формата.

LITERATURA

[1] Vikipedijini korisnici, "Internet stvari," Vikipedija, , //sr.wikipedia.org/w/index.php?title=%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82_%D1%81%D1%82%D0%B2%D0%B0%D1%80%D0%B8&oldid=23043878 (pristupljeno jul 30, 2021).

[2] Zennaro M., "Introduction to the Internet of Things", ITU, https://www.itu.int/en/ITU-D/Regional-Presence/AsiaPacific/SiteAssets/Pages/Events/2017/Nov_IOT/NBTC%E2%80%93ITU-IoT/Session%201%20IntroIoTMZ-new%20template.pdf (pristupljeno jul 30, 2021)

[3] Salazar C. et al, "Internet of Things-IOT: Definition, Characteristics, Architecture, Enabling Technologies, Application & Future Challenges" Research Gate, https://www.researchgate.net/publication/330425585_Internet_of_Things-IOT_Definition_Characteristics_Architecture_Enabling_Technologies_Application_Future_Challenges (pristupljeno jul 30, 2021)

[4] Bigelow S, "What is edge computing? Everything you need to know," https://searchdatacenter.techtarget.com/definition/edge-computing (pristupljeno 1.08.2021)

[5] Trilles, Sergio et al. “An IoT Platform Based on Microservices and Serverless Paradigms for Smart Farming Purposes.” Sensors (Basel, Switzerland) vol. 20,8 2418. 24 Apr. 2020, doi:10.3390/s20082418

[6] Sakovich N., 10 Best IoT Platforms for 2021, SamSolutions, https://www.sam-solutions.com/blog/top-iot-platforms/ (pristupljeno 05.08.2021).

[7] Guth, J., A Detailed Analysis of IoT Platform Architectures: Concepts, Similarities, and Differences., 2018 Springer-Verlag., https://www.iaas.uni-stuttgart.de/publications/INBOOK-2018-01-A-Detailed-Analysis-of-IoT-Platform-Architectures-Concepts-Similarities-and-Differences.pdf (pristupljeno 05.08.2021)

[8] Cannday B., THE FUNDAMENTAL IoT ARCHITECTURE, https://www.losant.com/blog/the-fundamental-iot-architecture (pristupljeno 5. 08. 2021.).

[9] Web sajt https://www.idglat.com/afiliacion/whitepapers/Whitepaper-IoT-Platform-Architecture-an-Enterprises-Design-Guide-v1.32.pdf?tk=/: (pristupljeno 05.08.2021)

[10] Wikipedia contributors, "NGSI-LD," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=NGSI-LD&oldid=1029335815 (accessed August 7, 2021).

[11] Web sajt https://whatis.techtarget.com/definition/monolithic-architecture (pristupljeno 13.08.2021)

[12] Web sajt https://microservices.io/patterns/monolithic.html (pristupljeno 13.08.2021)

[13] Web sajt https://microservices.io/patterns/microservices.html (pristupljeno 14.08.2021)

[14] Web sajt https://microservices.io/patterns/data/saga.html (pristupljeno 24.08. 2021)

[15] Web sajt https://kafka.apache.org/documentation/#gettingStarted (pristupljeno 24.08.2021)

[16] Web sajt https://spring.io/projects/spring-boot (pristupljeno 24.08.2021)

[17] Web sajt https://mosquitto.org/ (pristupljeno 24.08.2021)

[18] Wikipedia contributors, "InfluxDB," Wikipedia, The Free Encyclopedia, https://en.wikipedia.org/w/index.php?title=InfluxDB&oldid=1037003070 (pristupljeno August 24, 2021).

[19] Википедијини корисници, "MySQL," Википедија, , //sr.wikipedia.org/w/index.php?title=MySQL&oldid=23784099 (приступљено август 24, 2021).

[20] Web sajt https://www.ibm.com/cloud/learn/docker (pristupljeno 24.08.2021)

[21] Web sajt http://java-ml.sourceforge.net/ (pristupljeno 24.08.2021)