TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Allan Päll 193623IABB
Ürgen Sõukand 185528IABB

Asjade interneti seadmete haldamise lahendus koos automaatse avastusega Eesti Energia AS näitel

Bakalaureusetöö

Juhendaja: Tarvo Treier
Magister

Tallinn 2022

Autorideklaratsioon

Kinnitame, et oleme koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autorid: Allan Päll, Ürgen Sõukand

18.05.2022

Annotatsioon

Lõputöö eesmärk on luua lahendus, mis aitaks lihtsustada ettevõttes suure hulga IoT-seadmete süsteemi registreerimist ja nende edaspidist haldust.

IoT-seadmete registreerimise ning halduse muudavad keerukaks nende suur arv süsteemis ning seadmete toimimine erinevatel protokollidel. Lisaks sõltub seadmete kasutamise väärtus sellest, kas nad on seotud ettevõtte objektidega ning kuidas kasutatakse seadmetelt kogutud andmeid.

Selle probleemi lahendamiseks pakuvad autorid välja tervikliku süsteemi, mis avastab seadmed automaatselt ning lisab nad seejärel haldusportaali. See saavutati kombineerides olemasolevaid vabavaralisi tehnoloogiaid ja autorite enda kirjutatud koodiga.

Lõputöö arendati ettevõttes Eesti Energia AS. Lahendust kasutati varasemalt ettevõttes loodud laudade broneerimise prototüübi IoT-seadmetega. Projekti tulemusena avastatakse laudade juurde kuuluvad seadmed automaatselt, kui nad ühenduvad sõnumivahendajaga. Seejärel seadmete info salvestatakse ja lisatakse haldusportaali, kus on võimalik seadmed siduda vastava laua andmetega.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 60 leheküljel, viis peatükki, 28 joonist, kaks tabelit.

Abstract

SOLUTION FOR AUTOMATIC DISCOVERY AND MANAGEMENT OF INTERNET OF THINGS DEVICES BASED ON EESTI ENERGIA AS

The aim of this thesis is to create a solution that would make large scale deployment and management of IoT-device easier for enterprises.

Registration and management of IoT-devices becomes difficult because enterprises usually require a large number of devices for their operations. This is further complicated by use of different protocols for communication with and between devices. Further, to unlock economic value for enterprises, devices should be paired with an object they represent and about which they gather and transmit data. In addition, it is important that the collected data is processed in a useful way and is usable for taking actions.

To tackle these issues, authors propose a software solution that can discover IoT-devices automatically and then registering them to a management system. This was achieved by combining open-source projects and code written by the authors.

The thesis project was developed in Eesti Energia AS during an internship. Authors' solution as proposed was utilised with a previously developed office desk booking prototype system, which used E-Ink screens as IoT-devices.

The key result of the proposed solution is that IoT-devices are automatically discovered when they connect to a message broker, after which they are automatically added to the management system by the software. This enabled pairing them with a representing asset and creating rulesets based on telemetry the devices would later receive.

The thesis is in Estonian and contains 60 pages of text, five chapters, 28 figures, two tables.

Lühendite ja mõistete sõnastik

Alamseade	Seade, mis pole otseselt ühendatud internetiga, kuid on võimeline suhtluseks läbi juhtseadme.
Avastusskript	Kood, mis suudab tuvastada seadme lülitumist või väljumist võrgust varasemalt etteantud parameetrite järgi
Backlog	Nimekiri vajaminevatest arendusülesannetest, mis on. kirja pandud, kuid pole veel arendama hakatud.
CI/CD	*Continuous Integration / Continuous Deployment* - pidev integratsioon ning juurutamine on protsess, kus koodi integreeritakse peaharusse võimalikult tihti, kusjuures selle raames seda testides ning automaatselt juurutades.
Fork	Lähtekoodist tehtud koopia, millel alustatakse eraldiseisvat arendust
IoT	*Internet of Things* ehk asjade internet, eesti keeles tuntud ka kui nutistu, esemevõrk ja värkvõrk. Selle all mõeldakse internetiga ühenduses olevate seadmete võrku, kus seadmed on võimelised suhtlema omavahel, anda teavet ning täita ettemääratud reeglistike põhjal ülesandeid.
IoT-seade	Füüsiline seade, mis on ühendatud nutistu võrku, tekstis kasutatakse ka lihtsalt „seade“ nime all.
ISO	*International Organization for Standardization* ehk Rahvusvaheline Standardiorganisatsioon on riikide standardiorganisatsioone koondav rahvusvaheline organisatsioon.
Juhtseade	Seade, mis on otseselt ühendatud internetiga.
OASIS	*Organization for the Advancement of Structured Information Standards* on mittetulundusühing avatud standardite arendamise ja kasutuselevõtu edendamiseks.

ORM	*Object-Relational Mapping* on koodipõhise objekti ja andmebaasipõhise objekti teisendamise süsteem.
OSI	*Open Systems Interconnection model* on universaalne mudel, mis kirjeldab standardit telekommunikatsiooni ning arvutite suhtlemissüsteemidele.
Pull request	Tõmbetaotlus - algatus koodi lisamiseks teise koodirepositooriumisse või erinevate harude vahel.
Vara	Töö kontekstis tähendab vara objekti, mis on IoT-seadmega seotud, näiteks laud, ruum või hoone.
Webhook	Veebihaak ehk veebi kaudu teisele rakendusele tehtav käsklus, millelt oodatakse tegevust või tagasisidet.
Äärepealne jõudlus	Inglise keeles *edge computing* on arvutusjõudluse ja salvestusruumi kasutamine asukohas kohapeal seadmes.

Sisukord

Tabelite loetelu

Jooniste loetelu

1 Sissejuhatus

Ligikaudu 93% ettevõtetest kasutab mingil määral IoT-seadmeid [1]. Nutistu seadmed pakuvad mitmeid erinevaid võimalusi ettevõttele lisaväärtuse loomiseks. Näiteks parem kontroll varade üle, protsesside automatiseerimine või info korjamine, mille alusel andmeanalüüsi sooritada. Väärtuse loomiseks on lisaks IoT-seadmetele vajalik ka neid toetav arhitektuur ja tugisüsteemid. Sellisteks tugisüsteemideks on näiteks seadmete haldusportaalid, andmebaasid, analüüsi algoritmid ning tehnikud. Autorid keskenduvad süsteemi loomisele, mis toetaks IoT-seadmete registreerimise ja halduse aspekte. Süsteemi toimimist kirjeldatakse Eesti Energias meeskonna projekti raames arendatud laudade broneerimise süsteemi prototüübi näitel. Prototüüp kasutab IoT-seadmeid laudade broneeringute kuvamiseks ning haldamiseks.

1.1 Probleem

IoT-seadmete kasutamine tööstusharus muutub järjest populaarsemaks, aastal 2021 kasvasid ettevõtete kulutused IoT-süsteemidele 22,4%, koguväärtuseni 158 miljardit dollarit [2]. Seadmeid kasutatakse nii loenduritena kui ka süsteemide kontrollimiseks. See aga tähendab, et IoT-seadmete arv ühes süsteemis võib muutuda väga suureks. Lisaks on IoT-seadmetest maksimaalse väärtuse saamiseks oluline, et seadmed suudaksid omavahel suhelda ja oleksid ühenduses inventuuri-, klienditoe-, äri- ja analüütikasüsteemidega [3]. Sellise suuremahulise ja omavahel ühenduses oleva süsteemi loomise muudab aga keerukaks IoT-seadmete riistvara, kommunikatsiooniprotokollide ja andmemudelite mitteühilduvus.

Mitmete süsteemide puhul kasvab IoT-seadmete kasu koos andmepunktide kasvuga. Selle illustreerimiseks võib vaadelda temperatuuri mõõtmist – mida rohkem mõõtepunkte, seda täpsem on info. Kui temperatuuri hoidmine on ärikriitiline, siis seda ei saavuta mõne üksiku seadmega. Samas vaatavad ettevõtted ka seda, kas seadmete üles seadmise ning haldamisele kuluv töökoormus ja -kulu võib üles kaaluda seadmete kasutamisest saadava majandusliku kasu.

Eesti Energia näitel on peakontoris üle 600 kontorilaua. Kui ettevõtte otsustaks igale lauale ekraani või muu sarnase IoT-seadme panna, siis selle süsteemi seadistamine ning seadmete töökindluse tagamine oleks suure halduskoormusega. Olukorras, kus broneerimissüsteem oleks töötajate poolt igapäevaselt kasutuses, on oluline tagada seadmete töökindlus. Seda luues tuleb aga kaaluda, mis on seadmete haldamise kulu. Sama küsimus kulude kohta tekib ka muude IoT-seadmete laiaulatuslikul rakendamisel. Selle prototüübi alusel arendasid töö autorid välja võimalikult üldise süsteemi, mis võimaldaks lahendada probleemi laudade broneerimise süsteemi näitel, kuid oleks laiendatav ka teistsuguste kasutuseesmärkidega seadmetele.

1.2 Eesmärk

Lõputöö eesmärgiks on luua tarkvaraline lahendus, mis lihtsustaks ettevõttes suure hulga IoT-seadmete ja nende alamseadmete süsteemis registreerimist ja haldamist. Seda tehakse kontorilaudade broneerimise prototüübi edasiarenduse raames, et tagada selle süsteemi jaoks vajalike IoT-seadmete ülesseadmine.

Eesmärgi täitmiseks saadi sisendeid Eesti Energia töötajatelt, ettevõtte sisestelt lahendustelt kui ka juba eksisteerivate IoT-seadmete haldamiste rakendustelt. Agiilsete praktikate abil analüüsiti ning lahendati järgmiseid olulisemaid alamülesandeid:

- IoT-seadmete valiku analüüs ning erinevate suhtlusprotokollide kaalumine
- IoT-seadmete avastamise automatiseerimine
- IoT-seadmete töökindluse ning võrgus olemise staatuse automaatne jälgimine
- seadmete haldusplatvormide analüüs ning alternatiivide kaalumine
- võimalused siduda seadmete infot ettevõtte teiste infosüsteemidega
- terviklahenduse võimalikult lihtne seadistamine ning üles seadmine

1.3 Töö struktuur

Töö on jagatud kuueks sisuliseks osaks. Esimeseks on sissejuhatus, kus kirjeldatakse probleem ning töö eesmärk koos alamülesannetega. Teiseks osaks on metoodika, kus

kirjeldatakse arendatava objekti algseis ja räägitakse töös kasutatud arendustööriistadest ja meetoditest. Kolmandas osas kaardistatakse lahenduse nõuded ning kirjeldatakse nende väljakujunemist. Neljandaks on lahenduse tulemuste kirjeldus, kus on kirjeldatud üldist arhitektuuri ning seejärel igat arhitektuuri kihti süvitsi. Viiendas peatükis analüüsitakse lahendust tervikuna, selle sobivust ja tuuakse välja lahenduse tugevused ning nõrkused. Kuuendas osas asub kokkuvõte. Töö lõpus asub kasutatud allikate kirjeldus ning lisad, kus sisaldub ka autorite eneseanalüüs ja panus.

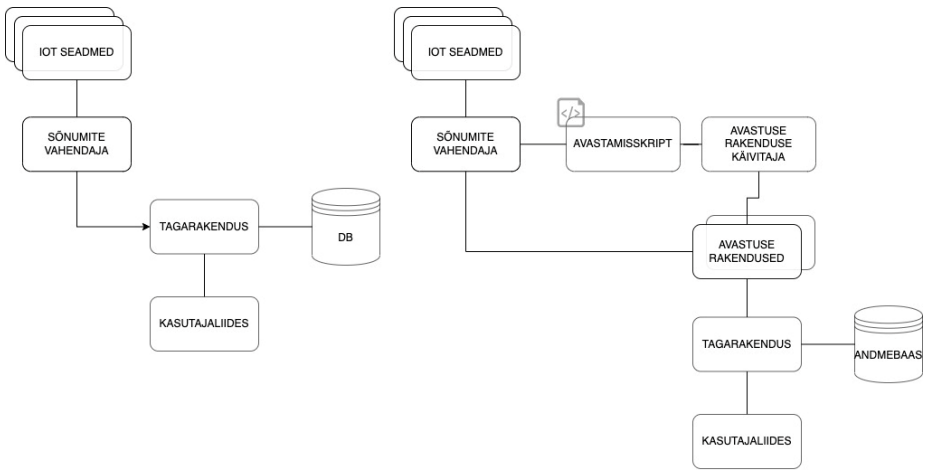
2 Metoodika

Selles peatükis kirjeldatakse, millised olid tarkvara väljatöötamisel kasutatud tööriistad. Lisaks antakse ülevaade kasutatud arendusmetoodikatest.

2.1 Objekti kirjeldus

Lõputöö projekti alguspunktiks oli töölaudade broneerimise süsteemi prototüüp, millel oli kaks peamist funktsionaalsust. Esiteks oli võimalus veebiliidese kaudu reserveerida töölauda ning seda muuta või tühistada. Teisalt loodi sõnumiedastuse kaudu võimalus kuvada ühele ekraanile, kas hetkel on sellega tinglikult seotud laud kinni või millal tuleb järgmine broneering. Prototüüp oli iseseisev ning ei olnud ühendatud ettevõtte süsteemidega nagu Active Directory või varahaldussüsteemidega.

Prototüübi komponentide üldine arhitektuur on joonise 1 vasakul poolel. Tagarakendus (ingl. *backend*) loodi Java Spring raamistikul mis kasutab relatsioonilist andmebaasi. Prototüübi kasutajaliides loodi Vue.js raamistikuga kasutades Typescript keelt. Seadmete mikrokontrolleris kasutati MicroPython skriptimiskeelt ning seadmetena oli kasutusel SyncSign-i tooted. Ekraanidega ehk seadmetega suhtlemiseks kasutati sõnumivahendajana Mosquitto Eclipse MQTT protokollil põhinevat vahendajat. Sellega ühendusid otse nii tagarakendus kui ka nutistu seade.



Joonis 1. Laudade broneerimise süsteemi prototüübi üldine arhitektuuri mudel (vasakul) ja lõpuprojekti mudel (paremal).

Joonise 1 paremal poolel on kujutatud lõpuprojekti juurde arendatud funktsionaalsust, mis tuvastab sõnumi vahendajaga ühenduva IoT-seadme ning registreerib selle tagarakenduses ja hakkab automaatselt edastama seadmete infot. Seadmed ning nende info jõuavad automaatselt kasutajaliidesesse, kus kasutaja saab luua endale sobiva töölaua, kus kujutada seadmetelt tulevat infot graafiliselt. Samuti saab kasutaja andmete põhjal luua automaatseid teavitusi.

2.2 Arendustööriistad

Projekti arendustööriistade valik tulenes ettevõttes kasutusel olevatest tehnoloogiatest. Arendustöös kasutati ettevõtte arvuteid ning võrku, mistõttu oli ka tarkvara laadimine arvutisse piiratud. Piirangud tulenesid ettevõtte turvapoliitikast, kus arvutisse sai alla laadida ainult auditeeritud programme.

Tarkvara kirjutamisel kasutati IntelliJ ning Visual Studio Code rakendusi. Koodihalduseks kasutati GIT-i ning koodihoidlana Githubi ning CI/CD protsesside jaoks Github Actions platvormi. Rakenduste koos käivitamiseks olid kasutuses Docker, Docker Desktop ning Azure Kubernetes Service (AKS).

Andmebaasile ligipääsuks kasutati pgAdmin kasutajaliidest ning andmebaas loodi PostgreSQL relatsioonilise andmebaasihaldussüsteemiga. Sõnumiedastuseks kasutati Mosquitto MQTT vahendajat ning selle jälgimiseks MQTT Explorer klientprogrammi.

Koodi testimisel kasutati IntelliJ ning Gradle koodiautomaatika tööriistu, mis võimaldasid automaatset koodi ehitamist, testimist ning koodi vastavust kokkulepitud stiilijuhendile. Samuti kasutati testimisel Postmani, kus kasutati testides Javascripti.

IoT-seadmete suuremahuliseks simuleerimiseks kasutati programmi nimega IoT-Data-Simulator. See võimaldas genereerida tuletisi olemasolevatest andmekogumitest, määrata sõnumite edastamise kiiruse ja sisu.

2.2.1 Raamistikud

Rakenduste arendamiseks kasutati Spring 5 raamistikku ning sellel põhinevat Spring Boot laiendust. Spring Boot eeliseks on lihtsustatud programmeerimismudel, mis võimaldab rakenduse seadistusi teha võimalikult lihtsaks, mistõttu sobib see hästi prototüüpimiseks.

Lisaks raamistikele kasutasid autorid lahenduses ka valmis rakendusi kui nende kasutamine aitas eesmärki täita ning see säästis tööhulka, mida lahenduse tegemiseks oli tarvis teha. Sellel põhimõttel võeti kasutusse ThingsBoard platvorm, mida kasutatakse peamiselt kasutajaliidesena, kui ka Akri, mis võimaldab rakendada automaatset riistvara avastust.

2.2.2 Programmeerimiskeeled

Tagarakenduses ning seadmete avastamise rakendustes kasutati Java JDK 11, pideva integratsiooni ning juurutamise (CI/CD) protsessides kasutati YAML keelt ning mikrokontrolleris Micropython keelt, mis on versioon Python 3-st ja on mõeldud kasutuseks vähese jõudlusega arvutites nagu mikrokontrollerid. Lisaks on kasutatud Rust programmeerimiskeelt, mida oli vaja Akri avastusskripti modifitseerimiseks.

2.2.3 Tehnoloogiad

Lahenduse tagarakenduse jaoks kasutati sõnumivahetuseks HTTP protokolli ning REST stiilil põhinevaid liideseid. Liideseid dokumenteeriti ning kirjeldati OpenApi ja SwaggerUI tööriistadega. Sõnumite edastamiseks seadmetega kasutati vahendajat, mis põhineb MQTT protokollil.

Lahenduse juurutamise testimiseks kasutati Azure pilves Kubernetest, mis võimaldab koordineerida Dockeri konteinereid omavahel ning tagada nende automaatne ülesseadmine. Kubernetese klaster käivitatakse koodireposiooriumist kasutates Github Actions käske (*webhooke).* Andmete talletamiseks kasutati relatsioonilist SQLi standardile lähedast vabavaralist andmebaasihaldussüsteemi Postgres ning selle haldamiseks PgAdmin rakendust.

Oluline oli tehnoloogiate, keelte ja raamistike valimisel tähele panna ja kaaluda nende litsentse ning võimalusel valida vabavara, mis lubab nii koodi muutmist kui kasutamist ettevõtluseks.

2.3 Arendusmetoodika

2.3.1 Projekti taust

Projekti arendati suuremas kuueliikmelises tudengite meeskonnas olles Eesti Energias praktikalepinguga tööl kaks päeva nädalas. Käesolev töö on jätk 2021. aasta sügissemestri

meeskonnatööprojektile, mille käigus valmis Eesti Energias töölaudade broneerimise süsteemi prototüüp. Põhinedes tagasisidele, mida saadi meeskonnaprojektile, otsustati projekt jagada sihtrühmade alusel kolmeks loogiliseks osaks: laudade broneerimise kasutajakogemus, ettevõtte ärivajaduste analüüs ning seadmete administreerimine.

2.3.1 Arendusprotsess

Arendusprotsessiks oli agiilne Scrum meetod, mida toetas ettevõtte poolt tooteomanik, Scrum Master ning arendajatest mentorid. Meeskond organiseeris ennast iseseisvalt ning alates jaanuarist kirjutati kasutajalood meeskonnaliikmete poolt ning projekti *backlog*-i haldasid ka meeskonnaliikmed vaheldumisi, kuigi Scrumis on see tooteomaniku ülesanne. Kuueliikmeline meeskond kasutas ühtset koodirepositooriumit ning toote *backlog*-i, kuid kõik piletid/kasutajalood märgistati lõputöö projekti märgisega ning neid haldas vastav tudengite tiim.

Järgiti kõiki Scrum artefakte nagu igahommikune püstijalakoosolek, pidev piletite täpsustamine (ingl. *refinement*), sprint, sprindi ülevaatus (ingl. *sprint review*), tagasivaatus (ingl. retroperspective) ning planeerimine (ingl. *sprint planning*) [4]. Sprindi pikkuseks oli kolm nädalat ehk kuus tööpäeva. Iga kahe tööpäeva tagant (igal esmaspäeval) tehti ka piletite täpsustamist, kus vaadati üle toote *backlog*, et tagada kõikide kasutajalugude selgus tiimile ning seati need prioriteetsuse järjekorda. Sprindi ülevaatus, tagasivaatus ning sprindi planeerimine tehti sprindi viimasel päeval üksteise järel.

2.3.2 Planeerimine

Backlog ülesanded kirjeldati veebruaris paaris ning nendele küsiti ettevõttest tooteomanikult ning mentoritelt üldist tagasisidet. Ülesanded jagati esialgu neljaks etapiks ning pandi olulisuse järjekorda. Ülesandeid hinnati ka nende kompleksuse poolest selliselt, et tiimiliikmed andsid oma hinnangu, kuid ilma et oleks teise hinnangut näinud enne kui neid üksteisele avaldati. Kui arvamused erinesid, tuli põhjendada ning hinnati uuesti kuniks jõuti ühise arusaamani ülesande keerulisuse osas.

Esimese sprindi jaoks kirjeldati ülesandeid detailsemalt ning hakati järjest võtma selliselt, et iga ülesande eest vastutas üks arendaja. Vajadusel küsiti üksteiselt ning ettevõtte mentoritelt nõu.

2.3.3 Koodi vastuvõtt

Lahenduse valmides laeti kood üles Githubi, hoides seda eraldi harus. Vastavalt kokkuleppele pidid enne peaharusse integreerimist kõik kuus üliõpilast muudatused üle vaatama ning heaks kiitma, misjärel andsid koodile vajadusel tagasisidet ka mentorid. Koodibaas oli lõputööde teemadel ühine, mistõttu integreeriti harud pärast tõmbetaotluse tagasisidestamist ning vajalike muudatuste tegemist peaharusse.

Koodi kvaliteedi ning rakenduse toimimise huvides oli kasutusel ka pidev integratsioon. Samuti loodi käsud, mis võimaldasid pidevat juurutamist valitud harudesse integreerimise järel. See seadistati selliselt, et iga kord jooksutati automaatteste. Pideva juurutamise kasutamine aitas olla kindlad kindel lahenduse toimimises ka konteineritena pilves ning andis kiiret tagasisidet, kui tekkis koodis tõrge, mis oleks takistanud mõnel rakendusel käivituda. Pideva integratsiooni ja juurutamise tööriistadeks olid Github Actions skriptid, Dockeri seadistusfailid ning Kubernetese keskkond ning selle jaoks tarvilikud konfiguratsiooni failid.

3 Nõuded

Selles peatükis antakse ülevaade tarkvara nõuetest, nende väljatöötamise protsessist ning põhjendused, miks sellised nõuded valiti. Tarkvara nõuete eesmärgiks on kirjeldada funktsionaalsust ja süsteemi toimimist. Üldjuhul jaotatakse tarkvara nõuded kaheks – funktsionaalsed nõudmised ja mittefunktsionaalsed nõudmised. Funktsionaalsete nõudmiste eesmärgiks on kirjeldada süsteemi funktsioonid kasutajale ning mittefunktsionaalsete nõuete eesmärk on täpsustada tarkvara toimimist üldisemalt.

3.1 Nõuete kujunemine

IoT-seadmete automaatse avastuse nõue tulenes Eesti Energia huvidest testida eelnevalt ettevõttes kontseptsioonina katsetatud võrku ühendunud seadmete avastuse võimekust. Seda sooviti edasi arendada IoT-seadmete registreerimiseks ning katsetada laudade broneerimise süsteemi prototüübi ühe edasiarendusena.

Analüüsides seda soovi jõudsid autorid järeldusele, et ainult IoT-seadmete automaatne avastus laudade broneerimise süsteemi ühe osana ei loo ärile piisavalt suurt väärtust. Esiteks oli oluline luua süsteem, mis oleks universaalne erinevate lahenduste ning seadmete jaoks ja oleks seadmete kasutamise eesmärgist lahutatud.

Lisaks leidsid autorid, et peamine kitsaskoht on seadmete haldamisega kaasnev töö ning ajakulu. Suurema väärtuse loomiseks on vajalik koondada seadmed haldusprogrammi, kus neid oleks võimalik siduda asukohtade või varadega, mida nad esindavad, jälgivad või kontrollivad. Selleks, et saada paremat ülevaadet, millised haldusprogrammi funktsionaalsused veel ärile kasulikud oleksid, saadi sisendit Eesti Energia digitaalse töökoha teenusehaldurilt, kes haldab koosolekuruumide broneerimise lahendust.

Eesti Energia peamaja koosolekute korrusel on iga koosolekuruumi sissepääsu juures ekraanina kasutusel tahvelarvutid, mis näitavad eesolevaid broneeringuid ning mille kaudu saab ka ruumi reserveerida. Selliseid seadmeid on 23 ehk suhteliselt väike arv ning kuna tegemist on tahvelarvutitega, on nad on oluliselt võimekamad kui traditsioonilised

IoT-seadmed. Siiski pakkus selline lahendus paralleele, mida tasus uurida. Näiteks leidsid autorid, et nende koosolekuruumide ekraanide haldamiseks kasutatav lahendus ei ole laiendatav suurele hulgale seadmetele. Koosolekuruumide ekraanidega seotud halduse ülevaate põhjal järeldasime, et lisaväärtust looksid automaatne teavitussüsteem seadmete olukorra kohta ning seadmetelt tulevate andmete visualiseerimine. Ekraanide halduskeskkonna kasutajaliidest hinnates ning teenusehalduri tagasiside põhjal järeldasid autorid, et vajalik oleks võimalus seadmeid haldusportaalis grupeerida, järjestada ning luua otsingufunktsioonid.

Nõuded vormistati piletitena koos kuueliikmelises meeskonnas kuhu olid lisaks kaasatud tooteomanik ja mentorid. Iga selline kasutajalugu sisaldas lühikest kirjeldust nõude kohta ning vajadusel tehnilisi kriteeriume, mis peavad olema täidetud, et lugeda kasutajalugu vastuvõetavaks. Need lisati *backlog*-i ning edasi täiendati neid kahekesi vastavalt lõpuprojekti teemale.

3.2 Eksisteerivate lahenduste analüüs

Selleks, et veenduda, kas lahenduse arendus on vajalik, koostasid autorid kõigepealt olemasolevate lahenduste analüüsi. Analüüsi valiti rakendused, mis osaliselt või täielikult täidaksid eelnevas peatükis kujunenud nõudeid. Lisaks andis olemasolevate lahenduste analüüs mõtteid arenduseks.

3.2.1 Mitte vabavaralised lahendused

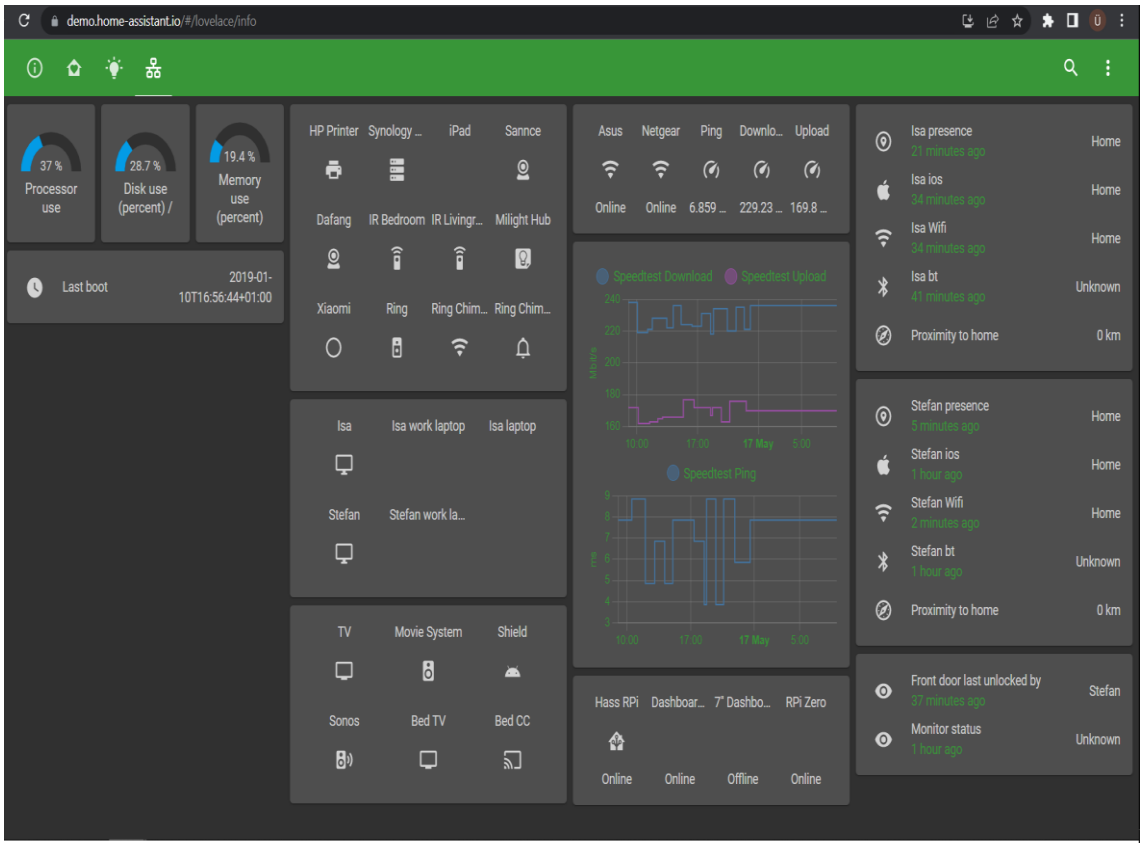
Suurimad pilveteenuste pakkujad on välja tulnud enda IoT-seadmete haldusportaalidega. Näiteks on Microsoft-il toode Azure IoT Central, AWS-il IoT Device Management, IBM-il Watson IoT Platform. Nende toodete kasutamine lukustaks haldussüsteemi ühe pakkuja teenusesse. Lisaks peab nende teenuste kasutamisel maksma kuutasu ning modifikatsioonid programmidesse tuleb tellida arendusena. Autorid otsustasid lisada mittefunktsionaalse nõude ainult vabavara kasutamiseks.

3.2.2 Home Assistant

Home Assistant on vabavaraline kodu automaatika kontrollimise rakendus. Esikohale on seatud privaatsus ning lokaalne kontroll [5]. Rakendus sisaldab mitut funktsionaalsust, mis kattuvad töös eelnevalt välja toodud nõudmistega. Näiteks on olemas automaatavastuse funktsionaalsus, kuid see töötab ainult seadmetega, mis on varasemalt

rakendusega liidestatud. See sobib hästi kodukasutajale, kes kasutab tuntumate tootjate IoT-seadmeid, kuid võib põhjustada probleeme tööstuslike IoT-seadmetega.

Lisaks on olemas MQTT tugi, mille abil telemeetria rakendusse tuua ning ka informatiivsete töölaudade loomise võimekus, kus neid andmeid kuvada (joonis 2).



Joonis 2. Kuvatõmmis Home Assistant platvormil loodud töölauast

Võimalik on luua ka reeglistikke automaatteavitusteks. Põhjus, miks see rakendus ei sobi välja töötatud nõuete täitmiseks, on puudulik varahalduse pool. Varasid, mida seadmetega siduda pole võimalik sisestada automatiseeritult. See on mõistetav, kuna rakendus on eelkõige mõeldud nutikodu loomiseks, kus on seotavateks varadeks erinevad toad, mida pole väga palju. Üheks võimaluseks oleks koodi *fork*-ida, et seda tööstusliku keskkonna vajadustele vastavaks muuta.

3.2.3 ThingsBoard terviklahendusena

ThingsBoard platvorm on mõeldud IoT-seadmete halduseks. Eksisteerib nii ettevõtetele mõeldud professionaalne versioon kui ka vabavaraline tarkvara. Autorid otsustasid kasutada vabavaralist varianti. ThingsBoard platvorm võimaldab kõiki soovitud

funktsionaalsuseid, kuid omab mõningaid probleeme ettevõtte kontekstis kasutamiseks. Nimelt puudub platvormil otse liidestamise võimalus ettevõtte teiste süsteemidega, näiteks varade haldusportaaliga või laudade broneerimise süsteemi prototüübiga. Lisaks ei pruugi ettevõtte otse liidestamist üldse soovida, kuna see võib muutuda turvariskiks, mille vältimiseks tuleks auditeerida kogu ThingsBoardi kood. Liidestamisega seotud probleemide tõttu otsustasid autorid ThingsBoardi terviklahendusena mitte kasutada. Kuid selle funktsionaalsuste ära kasutamiseks otsustati seda kasutada sarnaselt *front-end* rakendusele.

3.3 Funktsionaalsed nõuded

Lõpuks välja kujunenud süsteemi funktsionaalsed nõudmised olid järgnevad:

- Seadme lisamisel jälgitavasse võrku toimub selle seadme automaatne avastamine ja registreerimine haldussüsteemi
- Haldur saab luua, vaadata, muuta, kustutada seadmeid haldusportaalis
- Haldur saab luua seadme kategooriaid ning neid seadmetele määrata
- Haldur saab seadmeid kriteeriumide alusel otsida ning järjestada
- Lahendus võimaldab varasid automatiseeritult tuua haldusportaali, teistest varahaldussüsteemidest
- Haldur saab luua, vaadata, muuta, kustutada varasid haldusportaalis
- Haldur saab luua varade kategooriaid ning neid varadele määrata
- Haldur saab varasid kriteeriumide alusel otsida ning järjestada
- Haldur saab luua seose vara ja seadme vahel
- Haldur saab vaadata seadmeid, mis on varaga seotud/sidumata
- Haldur saab vaadata varasid, mis on seadmega seotud/sidumata
- Haldur saab luua reeglistiku mille alusel toimuvad automaatteavitused seadmete kohta

- Haldur saab luua seadmetelt saadud/saadavate andmete põhjal visuaalseid representatsioone
- Haldur saab teavituse kui uus seade on automaatselt registreeritud
- Haldur saab teavituse seadmete kohta, kui nad on olnud teatud aja võrgust väljas

3.4 Mittefunktsionaalsed nõuded

Lõpuks välja kujunenud mittefunktsionaalsed nõudmised olid järgnevad:

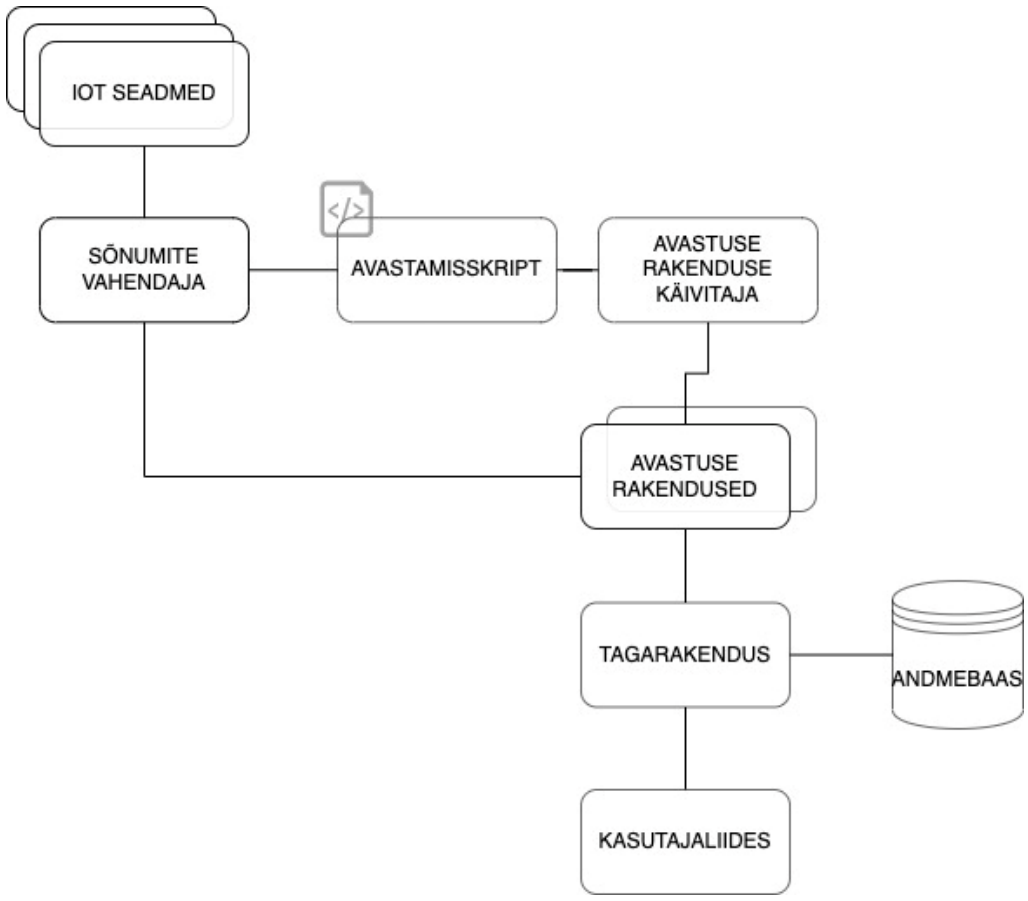
- Süsteem peab olema skaleeritav suure hulga seadmete jaoks
- Pidev integratsioon ja juurutamine
- Kõik tarkvara osad on võimalik paigutada konteineriseeritud keskkonda
- Järgitud on puhta ja kergesti muudetava koodi parimaid tavasid
- Võimalikult selgelt/lihtsalt laiendatav uut/erinevat tüüpi seadmetele
- Rakenduste liideste dokumentatsioon OpenApi põhjal
- Tagada info terviklikkus hajussüsteemis
- Valmisrakendused, mida lahenduses kasutatakse peavad olema vaba-varalised

4 Tulemused

Esmalt tutvustatakse arhitektuuri üldiselt ning seejärel liigutakse igasse arhitektuuri kihti süvitsi. Igas kihi kirjeldamisel alustatakse arhitektuurist ning lahenduse toimimise põhimõtetest, mida toetavad tehnilised joonised või väljavõtted koodist. Lugeja lihtsuse ja arusaadavuse jaoks tulemuste juures iga arhitektuuri kihi juures toodud välja ka selle kihi analüüs. Lahenduse kui terviku analüüsiga saab tutvuda järgmises peatükis.

4.1 Üldine arhitektuur

Autorid pakuvad välja hajussüsteemi, kus erinevaid lahenduse osi on võimalik üksteisest iseseisvalt arendada või teha ühe või teise komponendi juures valikuid, sealhulgas näiteks valmislahenduste täielikku või osalist kasutamist mõne komponendi jaoks. Joonisel 3 on välja toodud kogu lahenduse kõrgema taseme arhitektuur abstraktselt ilma konkreetseid tehnoloogiaid või lahendusi mainimata.



Joonis 3. Lahenduse arhitektuuri ülevaade

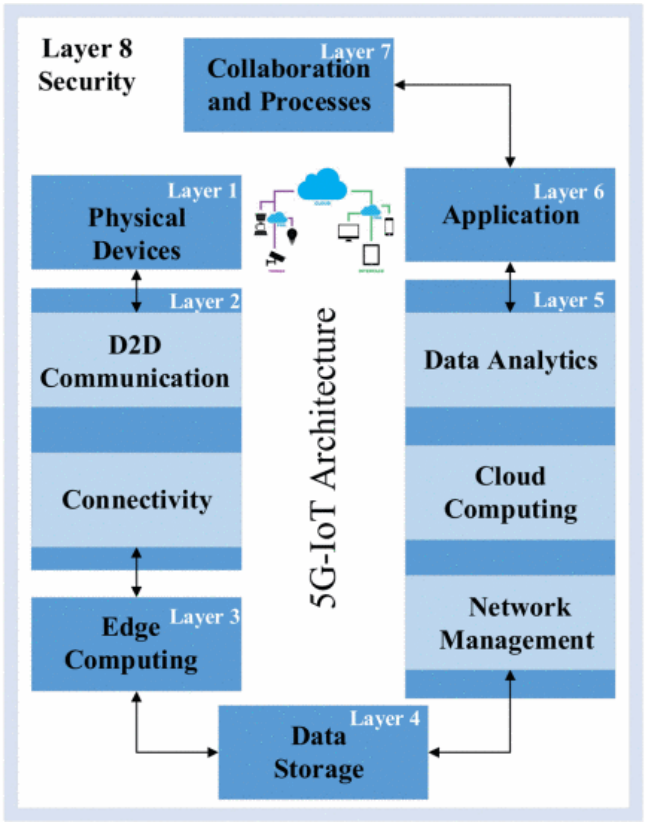
Lahenduses on 8 eraldiseisvat komponenti, nende põhiülesanded on:

- **IoT-seadmed** – koguda ja anda infot enda ümbrusele.
- **Sõnumite vahendaja** – võtab vastu sõnumid seadmetelt ning edastab need rakendustele või vastupidi.
- **Avastusskript** – jälgib muutuseid sõnumite vahendajas ja edastab need avastamise rakenduse käivitajale.
- **Avastamise rakenduse käivitaja** – käivitab avastamise rakenduse, kui talle kantakse ette muudatusest vahendajas.
- **Avastamise rakendus** – ühendub vahendajaga ning tellib sõnumid seadmele, registreerib sõnumite põhjal IoT-seadmed tagarakenduses ning edastab seejärel sõnumeid.
- **Tagarakendus** – tegeleb salvestavate andmete toimetamisega andmebaasi ning nende väljastamisega teistele rakendustele.
- **Andmebaas** – sinna salvestatakse info seadmete ja nende edastatud info kohta.
- **Kasutajaliides** – siin saab kasutaja andmetega suhelda ning teha toiminguid.

Lahenduse arhitektuuri ülesehitusel on põhinetud Rahimi, Zibaeenejad ja Safavi [6] väljapakutud arhitektuurile. Artikkel kirjeldab arhitektuuri 5G IoT-seadmete jaoks, kuid arhitektuur sobib ka antud lahendusele, sest arvestab selliste tehnoloogiate kasutamist nagu pilve- ning äärejõudlus ning on mõeldud erinevate andmetüüpide kasutamiseks.

Artikli alusel koosneb moodne IoT-arhitektuur 7 kihist – füüsiliste seadmete kiht, kommunikatsioonikiht, äärepealne arvutijõudluse kiht, andmesalvestuskiht, haldusteenuse kiht, rakenduse kiht ja koostöö ja protsesside kiht (joonis 4).

Lisaks tuuakse eraldi kihina välja ka turvalisuse kiht. Järgnevalt on kirjeldatud iga kihti süvitsi, kus tutvustatakse tehnoloogiaid mida vaatluse all olevas kihis lõpuprojekti raames rakendati.



Joonis 4. Väljapakutud 5G IoT arhitektuuri kihid. Allikas: [6]

4.2 Füüsiliste seadmete kiht

Füüsiliste seadmete kiht hõlmab endas kõiki süsteemiga ühenduses olevaid seadmeid. Nendeks võivad olla andurid, ekraanid ja juhtseadmed, kõik seadmed mis suudavad üle interneti suhelda või teha seda juhtseadme vahendusel. Süsteemi eesmärgiks on toetada võimalikult palju erinevaid IoT-seadmeid, et ettevõttel oleks valikuvabadus.

4.2.1 Kasutatud seadmed

Laudade broneerimise prototüübis olid kasutusel SyncSign Hub ja SyncSign E-Ink ekraanid (joonis 5), mida kasutati ka lõpuprojektis.

Joonis 5. Eesti Energia töölaudade broneerimise süsteemi prototüübis kasutatud IoT seadmed. Erakogu.

SyncSign Hub-i (joonisel 5 vasakul) saab ühendada võrku nii Wi-Fi kui ka Ethernet kaabli abil. Toite saab seade pistikust. See juhtseade toetab kuni 16 alamseadet, mis on konfigureeritud automaatselt ühenduma Zigbee protokolli alusel. SyncSign Hub ühendub ainult teiste SyncSign seadmetega ning selles jookseb tarkvara, mille kaudu on võimalik muuta seadistusi ning jälgida temaga ühenduses olevaid seadmeid ning nendele infot saata.

Integratsiooni jaoks ettevõtte süsteemi on SyncSign Hubi võimalik seadistada kasutades SyncSign pilveteenust REST liidese kaudu või osta litsents nende tarkvara kasutamiseks kohapealses serveris. Alternatiivina on võimalik juhtseadmel jooksutada Micropython koodi, mida ka prototüübi loomisel kasutati. Joonisel 6 on toodud välja kood MQTT vahendajaga liitumiseks ning tellitava ning publitseeritavate teemade konfiguratsioon.

```
# User App of Hub SDK, subscribe and publish MQTT messages

MQTT_HOST = "10.6.16.211"
MQTT_PORT = 1883
DEFAULT_KEEPALIVE = const(60)
KEEP_ALIVE_THRESHOLD = const(5)

def __init__(self, callback=None):
    self.subscribeCallback = callback

    self.sn = self.UNIQUE_ID
    self.client = None
    self.topicSubscribeOperation = self.sn + "/#"
    self.topicPubUpdate = self.sn + "/nodes"
    self.mqttLive = False
    log.info("PY: MQTT init")

def _clientInit(self):
    self.client = MQTTClient(
        client_id=self.sn,
        server=self.MQTT_HOST,
        port=self.MQTT_PORT,
        keepalive=60,
    )
```

Joonis 6. Seadmes käitav kood ühendumaks MQTT vahendajaga (Micropython).

E-Ink ekraanide näol on tegemist alamseadmetega, mis ise internetiga ei ühendu, vaid on ühenduses juhtseadmega raadiolainete abil. Kasutada oli 2,4 ning 4,2 tollise diagonaaliga ekraanid, viimastel on ka nuppude programmeerimise võimalus. E-Ink ekraane kutsutakse ka elektrooniliseks paberiks, kuna pildi kuvamiseks pole toidet vaja ning voolu kasutatakse ainult pikslite värvi muutmiseks, tehes nad energiasäästlikuks. Toite saavad ekraanid vahetatavatelt patareidelt.

SyncSign ekraanide pilt genereeritakse SyncSign tarkvaraga JSON formaadi alusel. Iga kuvatav element on JSON objekt, millel saab määrata asukohta ekraanil, suurust, tüüpi ning muid atribuute. See võimaldab erinevate mallide kasutamist erinevatel eesmärkidel ning JSON mallis vajalike andmete üle kirjutamist.

SyncSign Hub genereerib igale temaga ühenduvale alamseadmele identifikaatori, mille alusel on võimalik hiljem ainult sellele seadmele infot edastada. Laudade broneerimise prototüübi jaoks kirjutatud Micropython kood sisaldab erinevaid ekraanide JSON malle,

sõnumite vastuvõtmisel nende dekodeerimist baitide massiivist teksti, mallis teksti asendamist ning õige malli saatmist ekraanile, millele ta oli mõeldud.

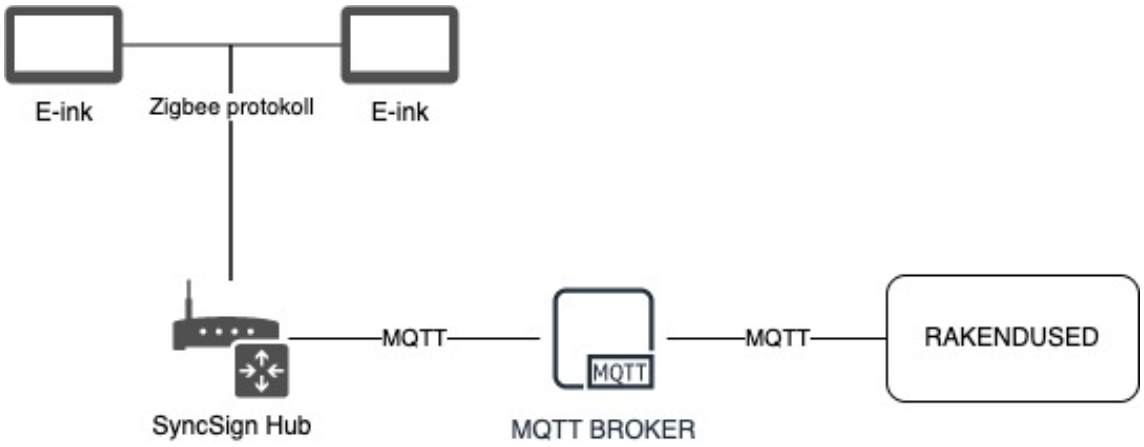
4.2.2 Füüsiliste seadmete kihi analüüs

Töös kasutuses olnud seadmed olid Eesti Energia poolt soetatud ning autorid neid ei valinud. SyncSign seadmed on ette konfigureeritud ning SyncSign pakub valmis liidestamise võimalusi läbi serveri tarkvara või nende pilveteenuse. See on hea lahendus kui ettevõte plaaniks ainult nende seadmeid kasutada. Samas tähendaks see, et seadmete ja nendega sisse seatud süsteemi ülalhoid pikas perspektiivis sõltuks sellest, millist tuge SyncSign pakub. Sellist sõltuvust soovib Eesti Energia võimalusel vältida ning võib eeldada, et see on oluline kriteerium paljudele ettevõtetele.

Kuna SyncSign seadmed võimaldasid ka oma tarkvara neil jooksutada, siis otsustati, et tuleb teha seadmete registreerimise süsteem, mis ei sõltuks konkreetsest seadme tüübist, aga tähtis on et seadmetele saaks laadida tarkvara. Selleks, et säästa aega tuleks erinevad seadmed valida selliselt, et neil saaks jooksutada sarnast tarkvara. Prototüübi rakenduse jaoks kirjutatud MicroPython skript ning selle muudatused projekti käigus on siiski sõltuvad konkreetsest seadmest. Kuid kui juurde soetada teisi seadmeid, mis suudavad ka MicroPython koodi jooksutada, siis oleks suur osa skriptist siiski üle võetav kuna meetodid on seal tehtud võimalikult üldised ning kood on dokumenteeritud.

4.3 Kommunikatsiooni kiht

Kommunikatsiooni kihi võib jagada kaheks - seadmete omavaheline suhtlus ning seadmetelt süsteemile suhtlus. Joonisel 7 on kujutatud suhtlust SyncSign seadmetega.



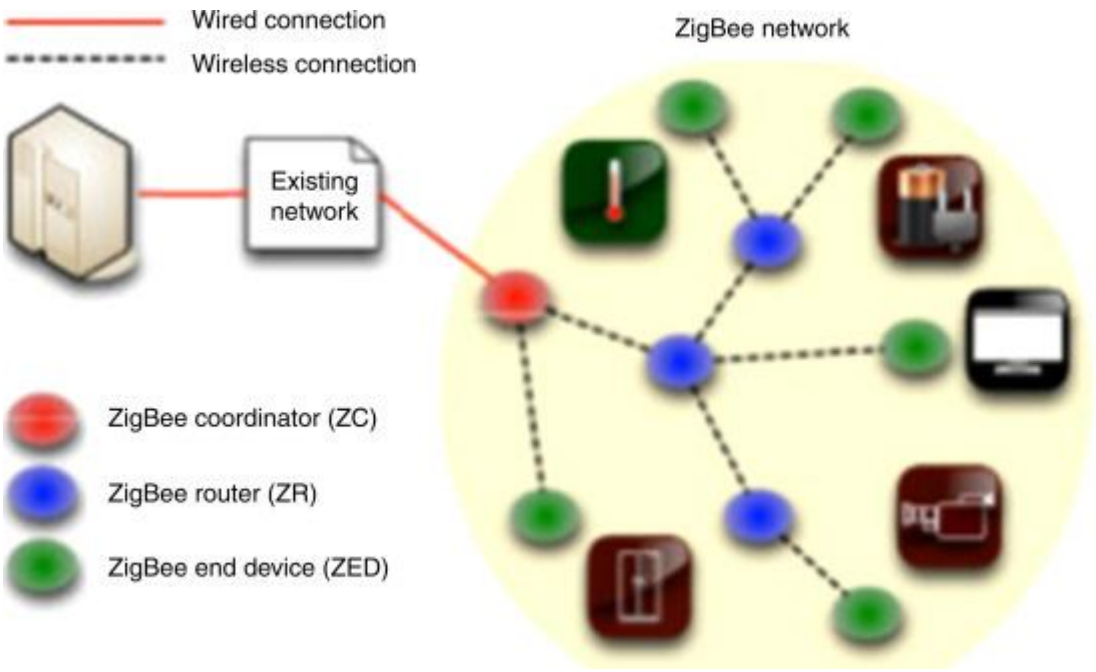
Joonis 7. Kommunikatsiooni kiht.

4.3.1 Seadmelt seadmele kommunikatsioon

Seadmelt seadmele kommunikatsioon toimub enamasti juhtseadmete ja alamseadmete vahel. On võimalik ka erinevate juhtseadmete vaheline suhtlus, kuid autorite lahenduses sellele ei keskenduta.

Hetkel kasutuses olevad seadmed suhtlevad omavahel Zigbee protokolli abil. Tegu on ISO/OSI mudelile ehitatud süsteemiga, mis rakendab ainult neid kihte, mis on vajalikud väikse voolutarbimise ja väikse andmeedastuskiiruse loomiseks. Võrk luuakse 868 MHz, 902-928 MHz või 2.4 GHz lainesageduse vahemikus. Sellised lainepikkused võimaldavad tavaliselt 10-100m kaugust andmeedastust ning andmeedastuskiiruseks on kuni 250 Kb/s [7]. See tähendab, et Zigbee protokoll on sobiv väiksemahuliseks andmeedastuseks, kuid tarbib võrreldes WIFI ja Bluetooth ühendustega palju vähem energiat.

Zigbee võrgu moodustavad raadiolained ja kolm erinevat tüüpi seadet (joonis 8). Esimeseks on Zigbee koordinaator, juhtseade, mis kogub infot lõppseadmetelt ja ruuteritelt ning tõlgib selle, et info välisele süsteemile edastada. Teiseks on ruuterid, mis võimendavad raadiolaine signaale, kuid võivad ka ise olla IoT-seadmed (näiteks nutipirn koos signaali võimendiga). Kolmandaks on lõppseadmed, mis ise võrku ei loo, kuid ühenduvad sinna, näiteks andurid, ekraanid, kaamerad jne.



Joonis 8. Zigbee võrgu ülesehitus. Allikas: Agarwal T. ZigBee wireless technology architecture and applications, elprocus.

4.3.2 Seadmelt süsteemi kommunikatsioon

Juhtseadmed ühenduvad kommunikatsioonikeskustega, et edastada enda info süsteemile. Oluline on, et kommunikatsioonikeskus oleks samuti skaleeritav, muidu võib tekkida kitsaskoht suhtluse edastamisel. Siin mängivad rolli nii ühenduvate seadmete ja saadetavate sõnumite arv, kui ka andmeedastuskiirused.

Lõpuprojektis on kommunikatsiooni protokolliks MQTT ja vahendajaks vabavaraline Eclipse Mosquitto. MQTT protokoll on OASIS standardi sõnumiedastus protokoll IoT-seadmete jaoks. See on disainitud võimalikult väiksemahuliseks ning töötab *publish/subscribe* meetodil. MQTT võimaldab kahesuunalist sõnumi edastust, kuni miljoneid ühendusi ühe vahendajaga, mida saab horisontaalselt skaleerida.

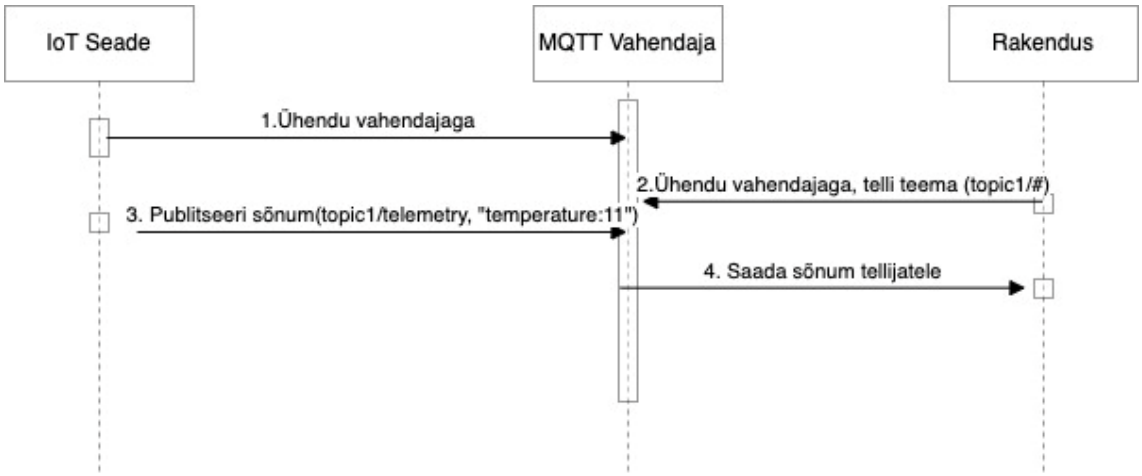
Toetatud on kolm erinevat kvaliteediastet sõnumite edastamise puhul: 0 ehk saada korra ja unusta , 1 ehk saada vähemalt korra ning 2 ehk saada ainult 1 korra [8]. MQTT sobib hästi ka võrkudes, kus ühenduse pidevus või kvaliteet pole ühtlased, mistõttu seab sätestada lahti ühendamiseks sõnumeid ehk „*last will and testament*“, et eristada korrapäraseid ühenduse katkestamisi nendest, mis juhtuvad näiteks võrguprobleemide tõttu. Autoris kasutasid kvaliteediastet 0 kuna tegemist pole ärikriitiliste andmetega ning võib lubada sõnumite kaotsi minekut, seda enam, et seadmete avastamine ja

registreerimine on lahendatud automaatika abil, millest tuleb juttu järgmistes peatükkides.

Autentimiseks saab määrata kasutajanime ning parooli, kuid lahenduse vahendaja seda ei nõua, et lihtsustada automaatse ülesseadmise protsessi. Vajalik on teada ainult vahendaja IP-aadressi. Selleks, et siiski jälgida turvalisuse protokolle, saab vahendajasse ühenduda ainult samast võrgust. Kuna enamikel ettevõtetel on loodud sisevõrk, mis on samuti autentimisega kaitstud siis peaks sellest piisama.

4.3.3 Tavapärane suhtlus MQTT protokolli abil

Publish/subscribe meetod on oma olemuselt üsna lihtne ja arusaadav. Nii IoT-seade kui rakendused ühendavad ennast MQTT vahendajaga. Kui klient soovib saada sõnumeid, saadab ta vahendajale *subscribe* sõnumi ehk „esitab tellimuse“, milles täpsustab, millist teemat ta kuulata tahab ning kui sellele teemale saabub sõnum, siis edastatakse see teema tellijale (joonis 9). Kui klient soovib vahendaja kaudu sõnumeid edastada, piisab kohe *publish* sõnumist, mis sisaldab nii teemat kui ka sõnumi sisu.



Joonis 9. MQTT tavapärane suhtlus.

4.3.4 Modifikatsioonid suhtluses automaatse avastuse rakendamiseks

MQTT suhtluse juures tuleb määrata teemad, millelt sõnumeid saadetakse/tellitakse. See on projektis lahendatud automaatselt, ehk klient teab enda identifikaatori alusel, mida tellida. Enamasti tuleb teemad defineerida klientrakenduse koodis ning kui seda tehakse käsitsi või kirjutatakse koodi sisse, võib see tähendada inimvigu või pidevat ümber

sätestamist. Mida rohkem seadmeid süsteemis on, seda tüütumaks muutub nende seadmete MQTT teemade seadistamine. Autorid kasutasid selle probleemi lahendamiseks modifitseeritud MQTT vahendajat, mida hakkab jälgima avastusskript.

Protsess näeb välja järgmine (joonis 10):

1. MQTT vahendaja koodi on muudetud selliselt, et vahendaja käivitudes luuakse teema "*brokerClients*". Seda teemat hakkab jälgima automaatse avastuse skript, mis ühendab ennast loomisel MQTT vahendajaga.

2. IoT-seade ühendub vahendajaga.

3. Lisatud on kood, kus MQTT vahendaja ise publitseerib sõnumi teemale "***brokerClients / {ClientId}***", kus *ClientId* on vastava kliendi identifikaator. Sõnumi sisuks on "1 ehk *online*".

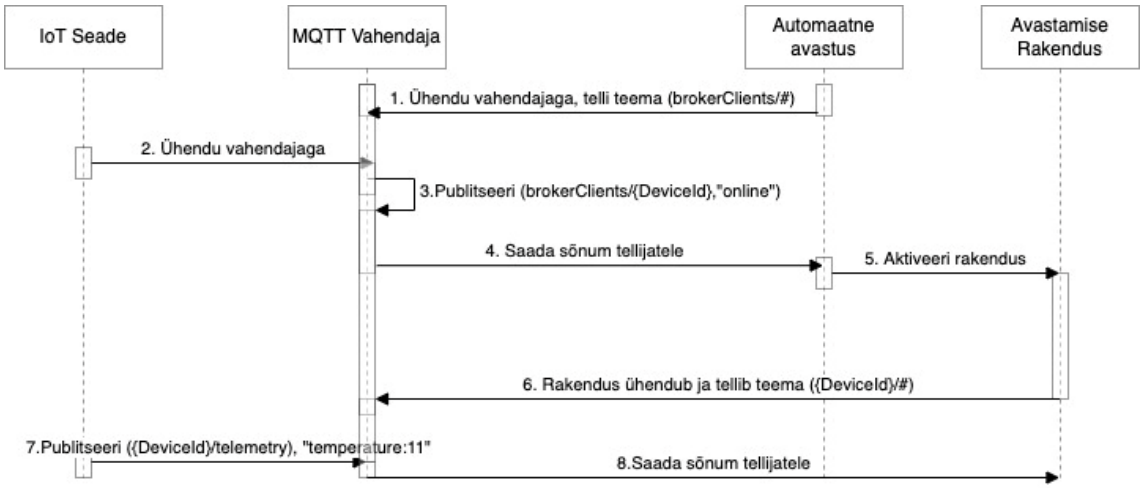
4. Kuna automaatse avastuse skript jälgib teemat "*brokerClients*", saadetakse eelmises punktis olev sõnum talle edasi.

5. Kui automaatse avastuse skript saab sõnumi, annab ta käsu klastris luua uus automaatse avastuse rakendus sellele kliendile, mis vahendajaga ühendus. Alamteemast teemast võetud *ClientId* sisestatakse keskkonna muutujana rakendusse. Iga klientseadme kohta luuakse oma rakendus, mis tõstab süsteemi töökindlust kuna võimaldab lihtsat skaleerimist.

6. Käivitatud rakenduses määratakse *ClientId* abil MQTT teema. Automaatse avastamise rakendus ühendub MQTT vahendajaga ning saadab tellimuse antud teemale. On määratus ka välistus, et avastusrakenduse ühendumisel MQTT vahendajaga ei tuvastata teda uue seadmena, mille jaoks oleks vaja uut avastusrakendust. Lisaks kasutatakse *ClientId* IoT-seadme registreerimiseks, kuid sellest on pikemalt juttu peatükis 4.4.

7. IoT-seade publitseerib telemeetriat teemale ***{ClientId} / telemetry***

8. Kuna rakendus kuulab teemat ***{ClientId}*** / # teemat, saadetakse talle telemeetria sõnum edasi.



Joonis 10. Modifitseeritud MQTT suhtlus rakenduste vahel.

4.3.5 Kommunikatsioonikihi analüüs

Seadmete vaheline kommunikatsioon

Projektis kasutatavad seadmed kasutavad omavaheliseks suhtluseks *Zigbee* protokolli. Hetkel on müüdud pool miljardit *Zigbee* kiipi ning aastaks 2023 ennustatakse selleks arvuks neli miljardit. Lisaks on turul üle 4000 sertifitseeritud *Zigbee* toote [9]. Seega võib järeldada, et *Zigbee* on üks populaarsemaid suhtlemisprotokolle ning toodete kasutamine on asjakohane.

Siiski tuleb ära märkida, et töös kirjeldatud süsteem ei sõltu sellest, millist protokolli seadmed omavahel suhtlemiseks kasutavad, vaid oluline on, et juhtseade tõlgiks sel viisil saadud info, et seda edastada üle võrgu, näiteks MQTT protokolli kasutamisel. Seadmete omavahelise suhtluse kirjeldus aitab mõista kogu süsteemi toimimist ja info liikumist tervikuna, kuid seadmete omavahelise suhtluse protokollist autorite pakutud lahendus ei sõltu.

Seadmelt süsteemi kommunikatsioon

Vahendaja kasutamine seadmete ning rakenduste vahel võimaldab vältida rakenduse sõltuvust mingitest konkreetsetest seadmetest ja vastupidi. Sõnumivahendaja teine oluline ülesanne on võimaldada suurel hulgal klientidel ühenduda ning vajadusel sõnumeid meeles hoida või järjekorda panna. Selliselt on võimalik tagada andmete edastamine asünkroonselt ning ka olukorras, kus ühendus ei ole pidev.

Sõnumiedastusprotokolliks valiti MQTT sest tegu on eelistatuima standardiga IoT suhtluses [10]. Lisaks vastab see OASIS ja ISO standardile. MQTT kasutamine on vähese nõudlusega ning toimib efektiivselt ka väikese edastuskiiruse või -mahu puhul [11].

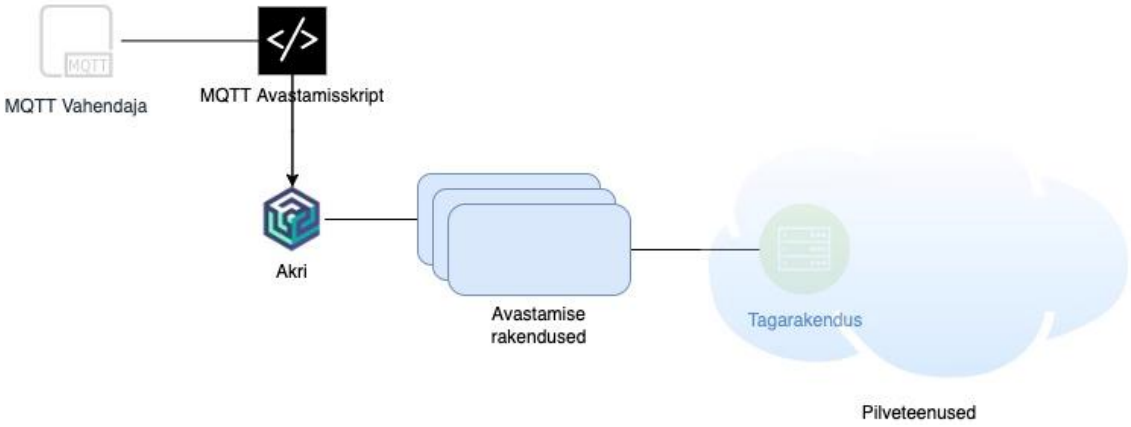
Mosquitto Eclipse vahendaja võeti kasutusele, kuna Eesti Energias oli selle lähtekoodist tehtud hargnemine (*fork*), milles oli koodi muutus. See muudatus võimaldas vahendajatega ühenduvatest klientidest MQTT teema abil teada saada. Seda võimalust kasutasid autorid automaatse avastuse funktsiooni loomiseks. Ühendatud klientide jälgimise teema ei loo mitte ainult aluse automaatse avastuse rakendamiseks, vaid võimaldab saata ka sõnumi, kui ühendus katkeb. Mosquitto Eclipsile on palju konkurente, näiteks HiveMQ ning EMQ X, kuid nende kasutamisel oleks pidanud ise kliendi haldamise implementeerima ning see oleks olnud ebapraktiline arvestades, et nad poleks pakkunud midagi olulist vajalikku lisaks.

4.4 Äärepealne arvutijõudluse kiht

Edge computing tähendab loogika täitmist füüsiliselt asukohas olevas seadmes. Selle eelis seisneb selles, et käske ning loogikat saab protsessida ilma interneti ühenduseta ning seeläbi ka kiiremini, kuna andmeid pole vaja edastada üle võrgu. Äärepealne jõudlus on siiski piiratud, kuna puudub suhtlus teiste seadmetega ja jõudluse mahud on piiratud kohaliku seadme jõudlusnäitajatega.

Töös kasutatakse äärepealset jõudlust seadmete automaatseks avastamiseks, kuna see võimaldab jälgida muutuseid äärepealses keskkonnas. Hetkel pakutud lahenduses monitooritakse muutuseid MQTT vahendajas. Seda oleks võimalik teha ka pilves, kuid kui tahaksime jälgida, millised USB IoT-seadmed ühendati arvutiga ning kasutada seda automaatseks registreerimiseks, poleks see pilvekeskkonnas võimalik.

Äärepealse jõudlusel toimival automaatavastusel on 3 põhilist komponenti (joonis 11) – avastusskript, vabavaraline programm Akri ja muudatuste avastamisel käivitatav programm. Need asuvad lokaalses Kubernetese klastris. Järgnevalt tutvustatakse neist igaühte.



Joonis 11. Äärepealse arvutijõudluse kiht.

4.4.1 Akri

Akri on Kubernetese ressursside liides, mis on loodud alamseadmete, näiteks IP-kaamerate ja USB-seadmete kujutamiseks Kubernetese klastri ressurssidena. Akri jälgib pidevalt seadme sõlmpunkte/ühendusi ning nendes muudatuste esinemisel käivitab eelkonfigureeritud koodi [12]. Kui Akri tuvastab, et ühendus katkeb, lülitatakse välja ka käivitatud kood.

Akri toetab hetkel kolme standardit seadmete avastuseks:

- ONVIF – protokoll kasutusel IP-põhistes turvatoodetes (kaamerad, andurid)
- OPC UA – suhtlusprotokoll tööstusautomaatika seadmete jaoks
- UDEV – seadmehaldur Linux masinate jaoks, haldab seadme sõlmpunkte /dev kaustas, näiteks mikrofonid, USB seadmed jne

Kuna tegemist on vabavaraga, saab toetatavaid standardeid lisada luues vastava standardi avastamisskripti.

Näide Akri kasutamisest: Objektile on vaja paigaldada IP-kaamerad turvalisuse jälgimiseks. Kaamerad kasutavad ONVIF protokolli. Ettevõtte võrku seatakse üles Linux server kaamerate halduseks. Serverisse installitakse lokaalne Kubernetes klaster ja seal seadistatakse Akri koos ONVIF protokolli avastusega. Nüüd uue kaamera ühendumisel ettevõtte võrku tuvastab Akri kaamera ning käivitab programmi, mis saadab kaameralt saadud pildi majahalduri jälgimisportaali.

4.4.1.1 Akri seadistamine

Akri kasutamiseks tuleb see esmalt seadistada. Kaks olulisemat komponenti selle juures on käivitatava programmi defineerimine ja avastusskripti konfigureerimine. Konfiguratsiooni fail on YAML keeles.

Käivitatava programmi konfiguratsioon on toodud joonisel 12. Sellega antakse Akrile teada, milline programm avastusel käima tõmmata. Selleks tuleb määrata repositooriumi üles laetud konteineri pilt (*image*). Lisaks on vaja kirjeldada käivitatava konteineri seadistus, näiteks kas konteinerit orkestreerimisel dubleeritakse (*capacity*) ning ressursid (*resources*), mis on konteinerile mälu ja jõudluse jaoks eraldatud.

```
brokerPod:
  image:
    # repository is the custom broker container reference
    repository: allanpall/desk-booking
    # tag is the custom broker image tag
    tag: device-discovery-1e9f55d6
    # pullPolicy is the custom pull policy
    pullPolicy: "Always"
  resources:
    # memoryRequest defines the minimum amount of RAM that must be available to this Pod
    # for it to be scheduled by the Kubernetes Scheduler
    memoryRequest: 10Mi
    # cpuRequest defines the minimum amount of CPU that must be available to this Pod
    # for it to be scheduled by the Kubernetes Scheduler
    cpuRequest: 45m
    # memoryLimit defines the maximum amount of RAM this Pod can consume.
    memoryLimit: 400Mi
    # cpuLimit defines the maximum amount of CPU this Pod can consume.
    cpuLimit: 500m
```

Joonis 12. Käivitatava programmi konfiguratsioon.

Avastusskripti nimetatakse konfiguratsioonis *discovery*, selle seadistus on toodud joonisel 13. Sarnaselt tuleb defineerida konteineri pilt, kus sisaldub avastusloogika ning repositoorium, kus seda hoitakse. Tuleb anda ka konteinerile kasutada antavad ressursid. Avastusskripti loogikast on põhjalikumalt kirjutatud järgmises peatükis.

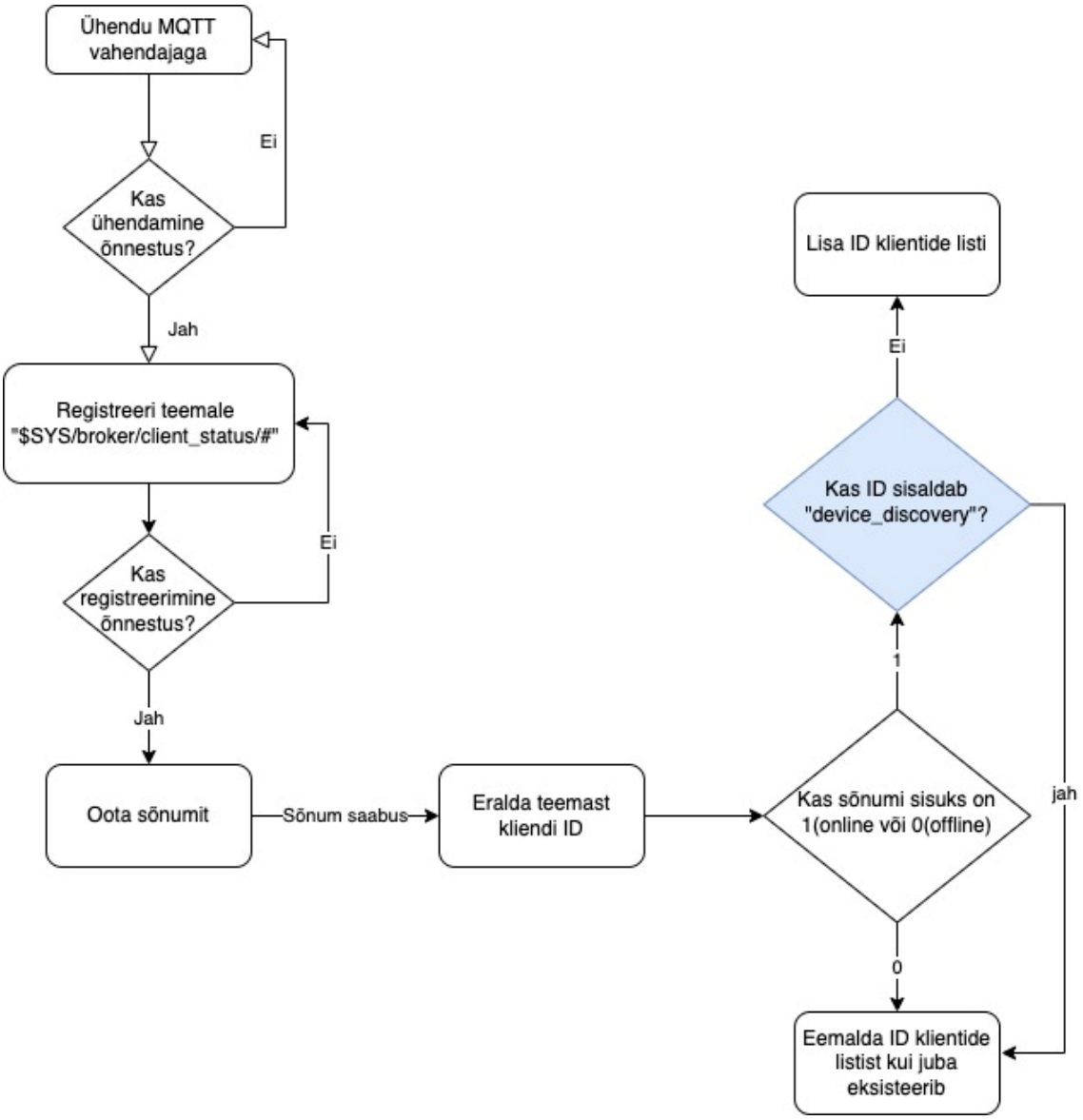
```
discovery:
  # enabled defines whether discovery handler pods will be deployed in a slim Agent scenario
  enabled: true
  # name is the Kubernetes resource name that will be created for this
  # custom Discovery Handler DaemonSet
  name: mqtt
  image:
    # repository is the custom broker container reference
    repository: allanpall/desk-booking
    # tag is the custom broker image tag
    tag: akri-discovery-handler-b056d4c4
    # pull Policy is the pull policy
    pullPolicy: "Always"
  # nodeSelectors is the array of nodeSelectors used to target nodes for the discovery handler to run on
  # This can be set from the helm command line using `--set custom.discovery.nodeSelectors.label="value"`
  resources:
    # memoryRequest defines the minimum amount of RAM that must be available to this Pod
    # for it to be scheduled by the Kubernetes Scheduler
    memoryRequest: 11Mi
    # cpuRequest defines the minimum amount of CPU that must be available to this Pod
    # for it to be scheduled by the Kubernetes Scheduler
    cpuRequest: 10m
    # memoryLimit defines the maximum amount of RAM this Pod can consume.
    memoryLimit: 24Mi
    # cpuLimit defines the maximum amount of CPU this Pod can consume.
    cpuLimit: 24m
```

Joonis 13. Avastusskripti konfiguratsioon.

4.4.2 MQTT avastusskript

Selleks, et Akri suudaks tuvastada muutuseid MQTT vahendaja juures, on selleks vajalik luua MQTT protokolli põhine avastamisskript. Kuna Eesti Energias oli varasemalt juba selline skript loodud, kasutati seda toorikuna. Avastamisskript on kirjutatud Rust programmeerimise keeles.

Joonisel 14 on sinisega märgitud loogika osa, mis lisati skriptile, et kontrollida, kas kliendi ID sisaldab sõna “*device_discovery*“, sest vastasel juhul oleks Akri poolt peale avastamist käivitatud rakendus samuti avastamisstandardi poolt leitud ning tekkinud lõpmatu avastamise ring. Seejuures on tagatud, et avastuse käigus käivitatav rakendus saab MQTT vahendajaga ühenduses oma identifikaatorisse samuti sõne „*device_discovery*“.



Joonis 14. Avastusskripti loogika.

4.4.3 Avastamise rakendus

Avastamise rakendus käivitatakse Akri poolt, kui avastamisskripti abil tuvastatakse uus MQTT klient. Projektis on avastamise rakendus kirjutatud Java programmeerimiskeeles raamistikule Spring Boot.

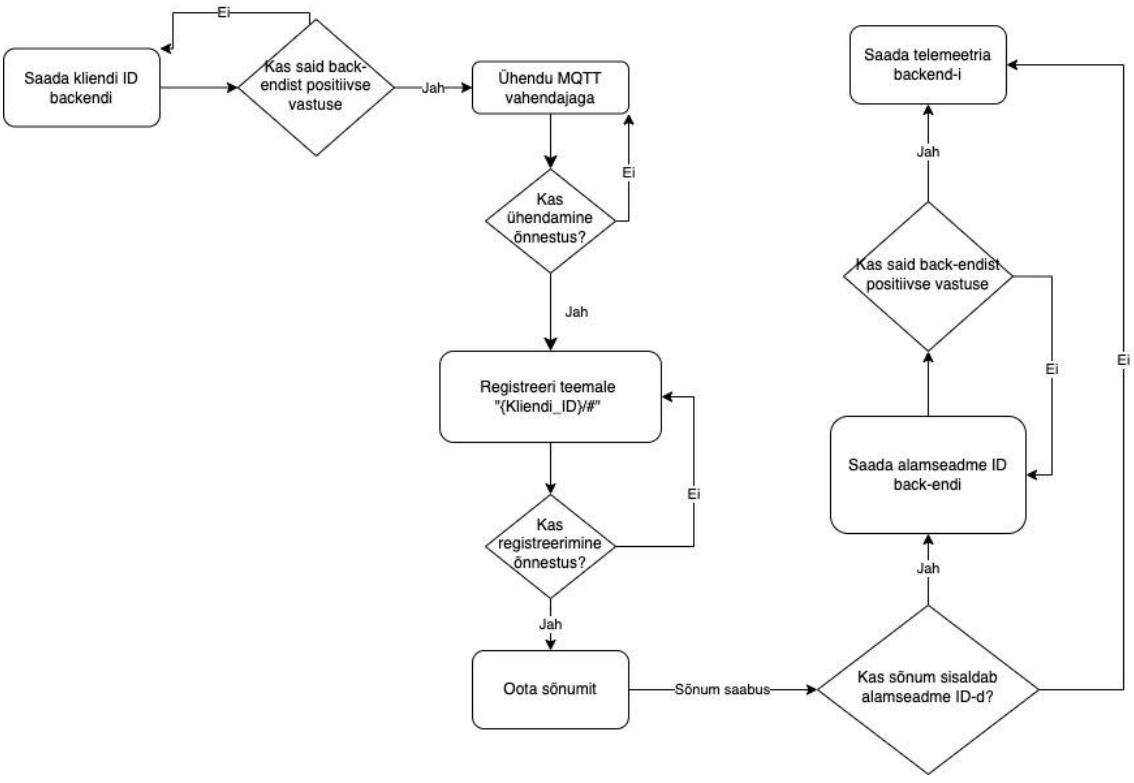
Rakendusel on kolm põhieesmärki:

- registreerida Akri poolt tuvastatud klient (IoT-seade) tagarakenduses,
- registreerida ennast kliendi teemale (*ClientId*) MQTT vahendaja juures,

- vahendajalt sõnumi saamisel tõlkida see ümber REST päringuks.

Joonisel 15 on toodud avastamise rakenduse toimimine. Rakenduse käivitamisel saadetakse koheselt HTTP POST-päring juhtseadme registreerimiseks tagarakendusse koos seadme ID-ga. Kui päringu vastuseks saadakse tagarakenduselt kood 200-OK, mis väljastatakse nii uue seadme registreerimisel kui juba olemasoleva seadme uuesti registreerimise katsel, ühendub rakendus MQTT vahendajaga.

Kui vahendajaga ühendumine on edukas, registreerib rakendus end kuulama teemat ***{Client_ID}*** / #, kus *Client_ID* on tema enda identifikaator ja „#“ tähendab, et kuula kõiki alamteemasid. Kui teemale saabub sõnum, otsutab rakendus, kas sõnumis sisaldub alamseadme identifikaator. Kui idendifikaator puudub, saadetakse tagarakendusse POST-päring telemeetria registreerimiseks, kus sisuks on juhtseadme ID ja telemeetria. Juhul, kui identifikaator tuvastatakse, tehakse tagarakendusse POST päring alamseadme registreerimiseks. Kui päringu vastuseks saadakse tagarakenduselt kood 200-OK, mis väljastatakse nii uue seadme registreerimisel kui juba olemasoleva seadme uuesti registreerimise katsel, tehakse päring tagarakendusse telemeetria registreerimiseks, kus sisuks on alamseadme ID ja telemeetria.



Joonis 15. Avastamisrakenduse loogika.

4.4.4 Äärepealse arvutijõudluse kihi analüüs

Olenevalt süsteemi suurusest või vajadustest on võimalik seda kihti seadistada tööle kas äärepealselt või näiteks ühes pilves. Näiteks oleks võimalik kogu avastuse süsteemi püsti panna igas kohalikus võrgus igas ettevõtte hoones. Lahenduse rakendused on üles seatud Azure pilves, kuid Kubernetese kasutamise abil on selle ümber seadistamine võimalik üsna lihtsalt.

Akri

Akri kasutamise eelised:

- vabavara – pole kulutusi tarkvara kasutamiseks, lähtekood on nähtav ja arusaadav
- vähe ressursinõudlik – ehitatud Rust programeerimiskeeles
- jälgib nii seadmete ühendumist kui ka eemaldumist
- rakenduse automaatse käivitamise protsess on juba loodud
- toetab mitut erinevat IoT-seadme protokolli, protokolle saab lisada luues vastava avastusskripti
- toetab mitme avastusskripti koos toimimist
- küllaltki lihtne ülesseadmine
- korralik dokumentatsioon
- IoT-seadmeid ei pea konfigureerima/muutma avastamiseks
- kasutatav nii äärepealsetes seadmetes kui pilvekeskkonnas

Akri kasutamise puudused:

- sõltuv Kubernetes konteinerite haldussüsteemist
- vajalik Rust keele oskus parimaks toimimiseks
- küllaltki uus tehnoloogia ning leidub vähe õppematerjale peale ametliku dokumentatsiooni

Nagu näha, leidub Akri kasutusel oluliselt rohkem tugevusi kui nõrkuseid ning nad on ka olulise kaaluga. Seega on autorite hinnangul selle kasutuselevõtt õigustatud. Lisaks puuduvad Akri lahendusel otsesed konkurendid, mistõttu sarnase funktsionaalsuse saavutamiseks tuleks ise luua programm, mis jälgib seadmete ühendumist võrku/sõlme ning seejärel kasutab Kubernetese liidest, et käivitada või seisata konteinereid. See oleks väga pikaajaline arendusprojekt ning nõuab väga häid teadmiseid nii jälgitava IoT-seadme protokolli kui ka Kubernetese kasutamisest.

MQTT avastusskript

Kuna MQTT avastusskript oli varasemalt Eesti Energia poolt loodud, analüüsivad autorid ainult lisatud koodi. Kontroll, kas kliendi identifikaator sisaldab sõna „device_discovery“ oli oluline, et vältida lõpmatut ringi, kus Akri avastab kliendi ning sellele kliendile loodud rakendus avastatakse Akri poolt ning protsess käivitatakse uuesti ning uuesti. Hetkel on kontroll implementeeritud lihtsa tingimuslausega ning väärtus koodi sisse kirjutatud. Paindlikkuse mõttes võiks tulevikus muuta koodi selliselt, et saaks konfiguratsiooni faili abil anda sisse listi väärtustest, mille puhul avastusskript Akrit ei teavita.

Avastamise rakendus

Avastamise rakendus on disainitud selliselt, et järgmist loogikat saaks täita ainult esimese õnnestumisel, milleks on seadme registreerimine tagarakenduses. See väldib üleliigseid päringuid tagarakendusse, mis lükataks andmebaasi reeglite tõttu niikuinii tagasi ning säästab selle arvelt jõudlust. Näiteks ei käivitata MQTT vahendajaga ühendumist enne kui seadme ID registreerimise kohta on saadud kinnitav vastus. Sellise lähenemise puhul on oluline, et oleks määratud ka loogika uuesti proovimise reeglid. Pole mõistlik lõputult päringuid teha, sest see muutub samuti tarbetuks jõudluseks. Nende põhimõtete rakendamine on autorite arvates avastamise rakenduse juures oluline. Üheks edasiarenduseks päringute vähendamiseks oleks registreeritud alamseadmete rakendusse meeldejätmine, et ei peaks iga kord ootama nende registreerimise kohta kinnitust.

Kuigi päringute vähendamisega säästetakse jõudlus- ja mälumahte on üheks suurimaks kitsaskohaks praeguse avastusrakenduse juures rakenduse üldine ressursi nõudlus. Rakendus on kirjutatud Java programeerimiskeeles Spring Boot raamistikule, mis pole kõige sobivam kergete mikroteenuste loomiseks. Kuna rakendus käivitatakse iga avastatud IoT-seadme jaoks, on oluline, et tarbitavad ressursid oleksid minimaalsed.

Autorid valisid algselt Spring Boot raamistiku, kuna sellega saab väga lihtsalt ja kiirelt toimiva rakenduse luua. Tulevikus tuleb avastamise rakendus kindlasti üle viia mõnele väikse nõudlusega raamistikule ning keelele.

Teiseks suureks puuduseks avastuse rakenduse juures on hetkel seadmelt vastuvõetava sõnumi kindlaksmääratud struktuur. See struktuur sobib Eesti Energias projekti raames kasutatavate seadmete avastamiseks ja telemeetria edastamiseks, kuid ei suuda hakkama saada sellest oluliselt erinevate sõnumiformaatidega. See loob olulised piirangud süsteemi automaatse avastuse võimekusele. Kindlasti tuleb rakendus tulevikus ümber kujundada selliseks, et IoT-seadmelt saadava sõnumi struktuuril poleks avastamise ja telemeetria edastamisel tähtust.

Viimast saaks saavutada, et tagarakenduse RESTful liidese kaudu telemeetria objekt salvestatakse andmebaasi dokumendina ehk JSON objektina. Siis pole tähtsust, millisel kujul või milliseid täpseid andmeid mõni konkreetne seade edastab ning avastamise rakendusest saab liides, kus iga seadme tüübi kohta tõlgitakse vastav sõnum üldisele kujule, mida tagarakenduse liides nõuab.

4.5 Andmesalvestuskiht

Asjade interneti seadmete puhul on levinud, et seadmete telemeetria salvestatakse enne muid toiminguid sarnaselt logidena. Seda näeb ette ka Rahimi, Zibaeenejad & Safavi pakutud arhitektuur [6]. Andmete salvestuskihi eesmärk on tagada andmete korrapärane talletamine, mis saavutatakse läbi valideerimise, mida teeb tagarakendus.

Seadme avastamise järel toimub tema registreerimine ning andmete edastamine veebirakenduse kaudu, mis salvestab avastatud seadmed andmebaasi. Rakendus kasutab REST tehnoloogial põhinevat veebiliidest, mis on dokumenteeritud Open Api standardit kasutades. Automaatse avastuse ning andmete edastamise jaoks on kolm päringuaadressi, mis kõik nõuavad autentimist ning kontrollivad antud sisendi valiidsust.

4.5.1 Rakenduse API

Rakenduse liides on koodis kirjeldatud Swagger tööriista abil, mis loob automaatselt ka kasutajaliidese, mille kaudu saab teha rakendusse päringuid (joonis 16).

POST päring äärepealse (*edge*) seadme registreerimiseks

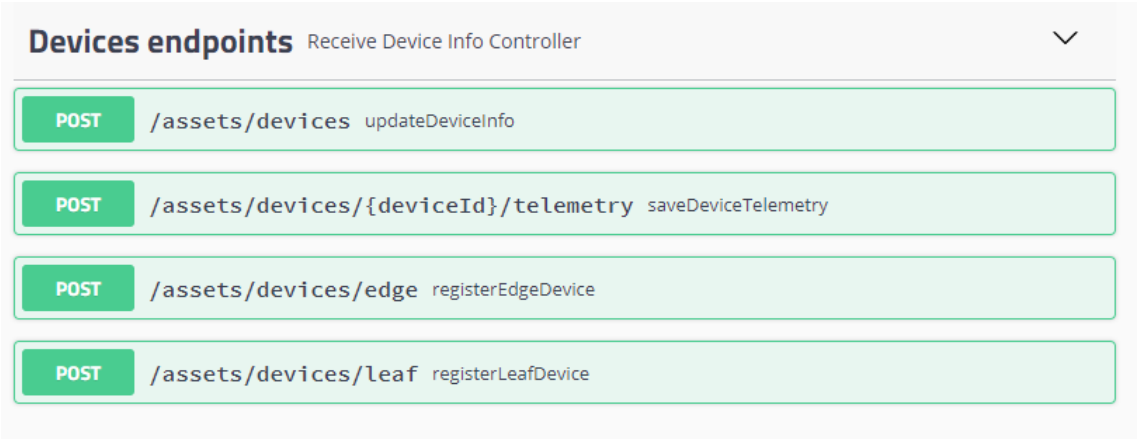
Õnnestunud päring salvestab uue seadme seadmetüübina Edge ning loob talle nime ning tagastab staatuse Ok (200). Kui saadetakse juba andmebaasis oleva seadme ID, tagastab samuti Ok (200), mis on vajalik näiteks juhul kui seade on vahepeal olnud võrgust väljas ning avastuse kaudu saadetakse uus päring.

POST päring alamseadme (*leaf*) registreerimiseks

Õnnestunud päring nõuab seadme IDle lisaks tema äärepealse seadme ID väärtust. Õnnestunud päring salvestab uue alamseadme ning seob selle vastava äärepealse seadmega. Kui ei õnnestu leida äärepealset seadet andmebaasis, tagastab vea *Not Found* (404) vastava sõnumiga.

POST päring seadmete poolt kogutud andmete edastamiseks

Õnnestunud päring nõuab, et seade mille ID on päringuga kaasas on juba andmebaasi salvestatud. Sestap tuleb kliendil enne andmete edastamist registreerida kas äärepealne seade või alamseade.

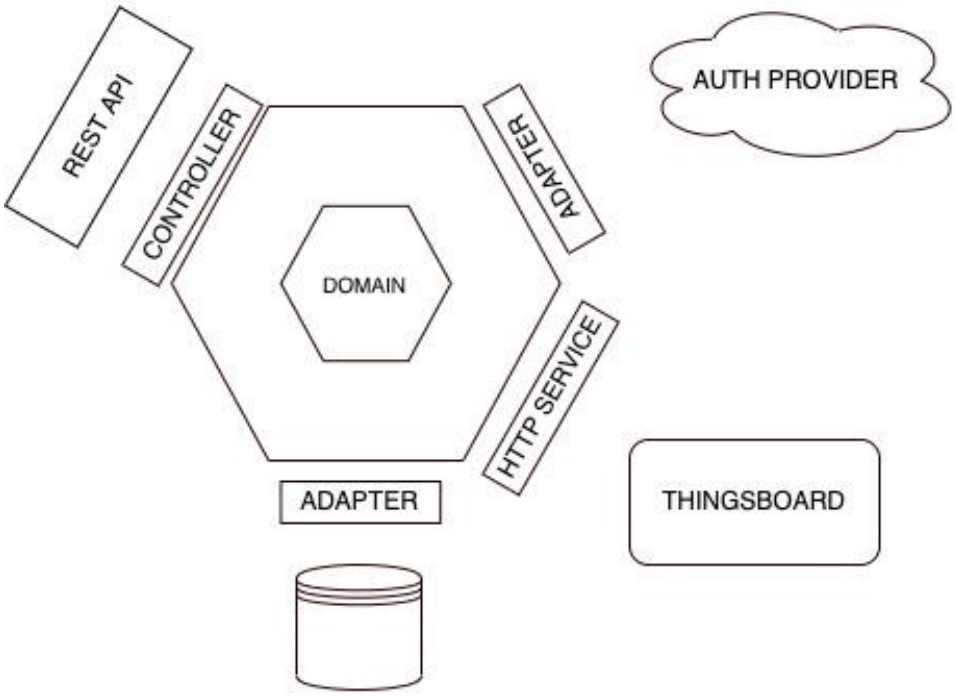


Joonis 16. Tagarakenduse liides OpenApi kasutajaliidese vaates.

4.5.2 Arhitektuur

Andmesalvestuskiht koosneb Spring Boot Java tagarakendusest ning relatsioonilisest andmebaasist. *Backend* rakenduse arhitektuur on kuusnurkne, mis põhineb puhta arhitektuuri põhimõttel, kus koodi sõltuvused on ainult ühes suunas ehk valdkonnamudeli suunas [13]. Rakenduse ülesehitusel järgiti Tom Hombergsi avatud koodirepositooriumi, mis kirjeldas kuusnurkse arhitektuuri kasutamist Spring raamistikuga, kusjuures tagades

arhitektuurimustrist kinnipidamise vastavate testidega [14]. Joonisel 17 on toodud ülevaade, kuidas rakendus on liidestatud erinevate kolmandate teenustega, kusjuures käskude jada liigub alati enne keskele ehk domeeni ning siis sealt liideste kaudu rakendusest välja.



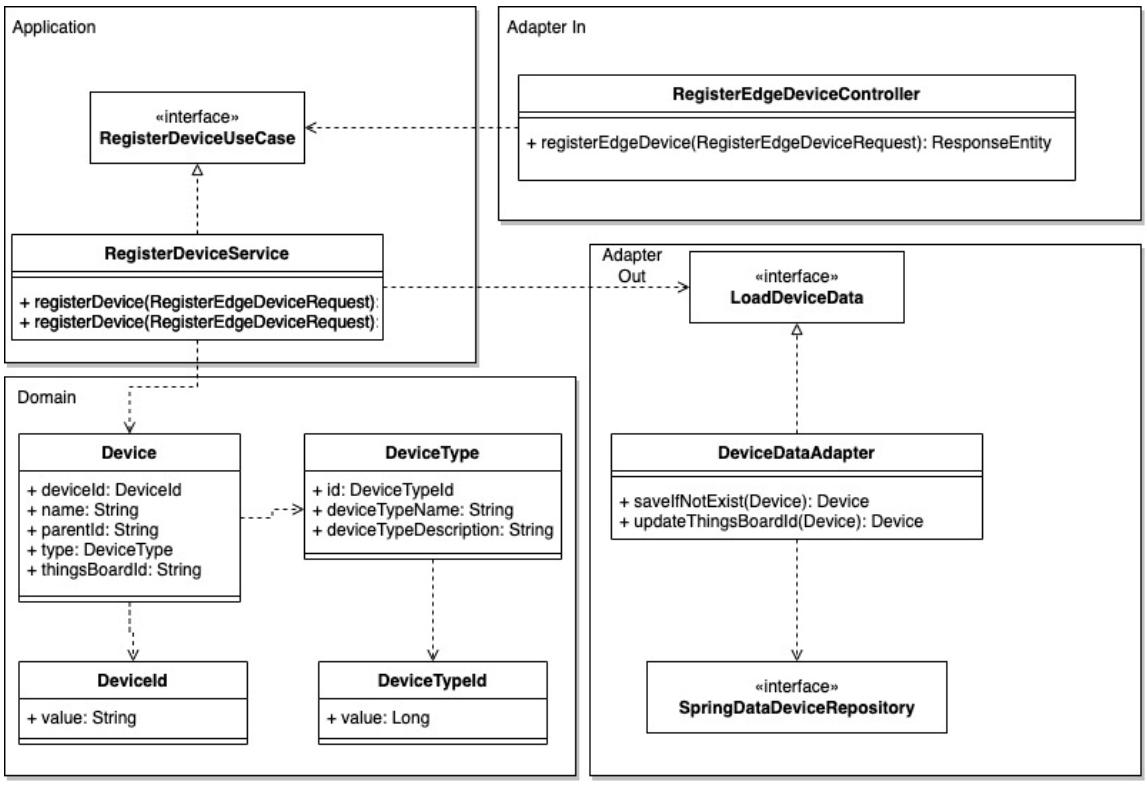
Joonis 17. Kasutatud kuusnurkse arhitektuuri mudel.

Kõige keskel on ärivaldkonna põhine kood, mis põhineb valdkonnamudelil (*domain*) ning sisaldab ärireegleid, mis väljenduva klassi invariantidena. Valdkonnamudel ei ole koodis sõltuvuses ühestki teisest rakenduse osast või konkreetsest infrastruktuurist. Selline sõltuvus on saavutatud ehitades ärivaldkonna kihi ümber rakenduse sisemiste teenuste ning väliste adapterite kihid. Kõigi kolme kihi vahel on liidesed, mis pööravad sõltuvuse ümber ja võimaldavad muuta rakenduse erinevaid osasid üksteisest sõltumatult.

4.5.3 Valdkonnamudel

Rakenduste koodis järgiti põhimõtet, et kood kirjeldaks valdkonda ning et valdkonnamudel oleks koodi ning lahenduse arhitektuurist sõltumatu [13]. Sellise mudeli eesmärk on tagada probleemvaldkonna jaoks vajalike reeglite kirjeldamine ilma, et ta sõltuks tema kasutajatest või klientidest. Domeenipõhise disaini põhimõte on, et läbi liideste ning sõltuvuste ümberpööramisega tagatakse see, et ärireegleid on võimalik arendada ning muuta iseseisvalt teistest rakenduse koodi osadest, mis tegelevad kas andmete säilitamise, sõnumite vahendamise või kasutajaliidese probleemidega.

Joonisel 18 on näha kuidas valdkonnamudeli klassid ei sõltu klassidest, mis tegelevad rakenduse suhtlus- või salvestusfunktsioonidega. Lihtsuse mõttes on jooniselt välja jäetud klassid, mis tegelevad mudelite tõlkimisega klasside vahel. Erinevate rakenduse osade vahel asuvad liidesed võimaldavad defineerida lepingud klassi jaoks, mis võimaldab lepingu täitmiseks teha muudatusi ilma, et peaks muutma koodi selle kasutajas.



Joonis 18. Seadmete klassidiagramm.

4.5.4 Andmebaas

Andmebaasina kasutatakse relatsioonilist SQL andmebaasi, mis on loodud PostgreSQL andmebaasihaldussüsteemi kaudu. Arenduses kasutati andmebaasi loomiseks Dockeri pilti, lisaks juurutati andmebaas Azure Kubernetes Cluster pilveteenuses Github Action-ite abil. Andmebaasi initsialiseerimiseks, tabelite rakendusega ühenduse loomiseks kasutatakse Spring Boot JPA Hibernate, mis on Object Relational Mapper (ORM).

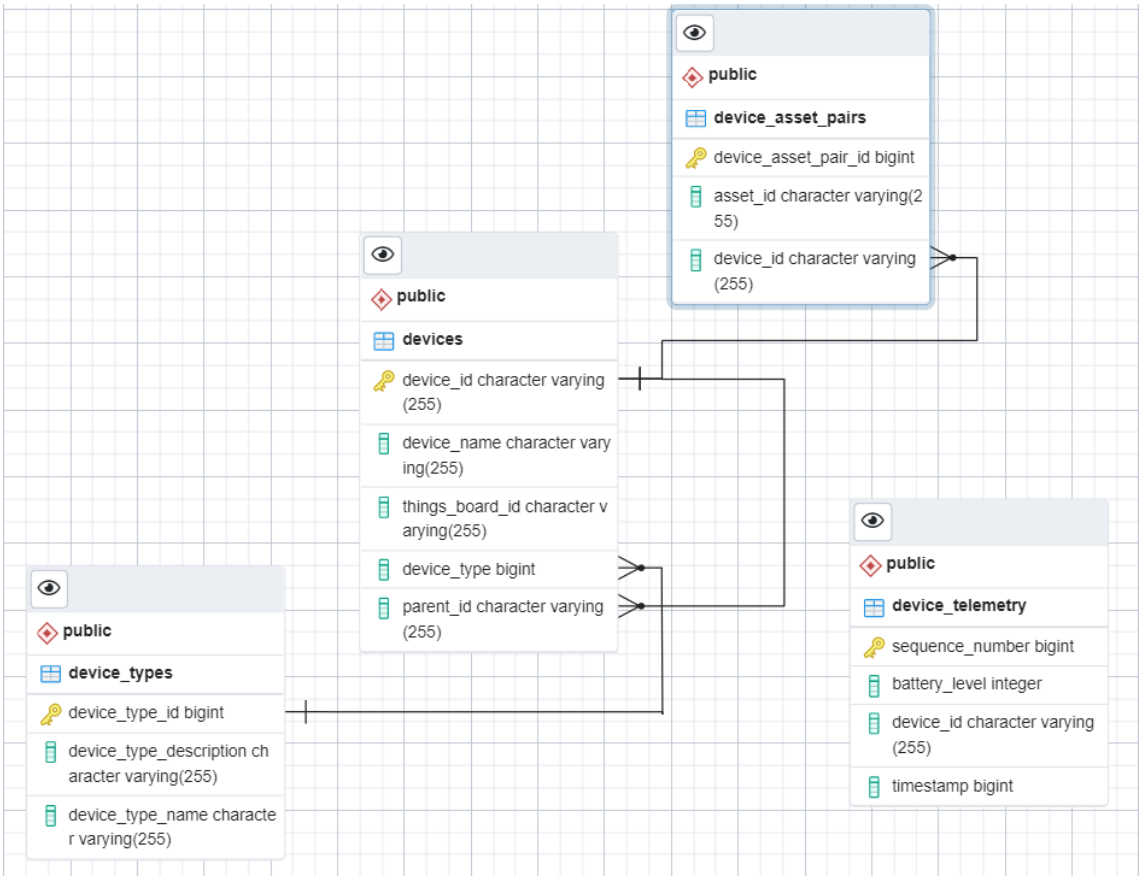
Andmebaasitabelid kood enne põhimõttel. Selleks on loodud eraldi klass iga olemi jaoks, kus on koodi annotatsioonidena kirjeldatud iga tabeli piirangud ja tema seosed teiste tabelitega. Tagarakenduse rakenduse koodis toimub andmebaasiga suhtlus ainult läbi liideste, kus on iga vajaliku kasutusjuhu kohta loodud vastav meetod. Igat liidest rakendab adapter, kus meetodid kasutavad andmete salvestamiseks Spring JPA repositooriumeid.

Andmebaasis koosneb neljast tabelist (joonis 19). Esimeseks on *devices* tabel, kus sisalduvad registreeritud IoT-seadmed. Defineeritakse seadme identifikaator, nimi ning tüüp. Juht- ja alamseadmete hierarhia sätestatakse kasutades veergu *parent_id*, mille kaudu alamseadmed seotakse juhtseadmega. Veerg *things_board_id* on sama seadme identifikaator Thingsboard keskkonnas ning seda kasutatakse telemeetria edastamiseks.

Tabel *device_types* on klassifikaator seadmete jaoks, seda kasutatakse näiteks defineerimaks, kas tegu on juht- või alamseadmega.

Tabel *device_asset_pairs* on loodud seadmete ja varade sidumiseks. Hetkel varade tabelit andmebaasis ei ole, kuid valmidus sidumiseks on sellegipoolest olemas.

Tabelisse *device_telemetry* salvestatakse telemeetria, mis seadmetelt saabub. See ei ole *devices* tabeliga välisvõtme abil seotud, kuna seda kasutatakse ka seadmetelt saadud info logina ning on seetõttu täiesti eraldiseisvaks tehtud. Lisaks on tulevikus plaan *device_telemetry* tabeli info salvestada hoopis JSON kujul, potentsiaalselt teise andmebaasi.



Joonis 19. Andmebaasi kasutajaliidese pgAdmin poolt genereeritud olemi-suhte diagramm

4.5.5 Testimine

Tagarakenduse konkreetseid funktsionaalsusi testiti nii ühiktestide, integratsioonitestide kui ka musta kasti viisil API testimisega. Lisaks kasutati ühiktestide automaatseks jooksutamiseks Gradle „*build*“ automatsiooni tööriistu ning kontrolliti automaatselt iga ehitamise käigus Gradle abil koodi stiili vastavust Google Java Style Guide nõuetele.

Ühiktestidega kontrolliti meetodeid isoleeritult ning neid jooksutati lokaalselt kui ka Githubis pideva integreerimise abil. Lahenduses kasutatud integratsioonitestid keskendusid rakenduse suhtlusele andmebaasiga, mida kontrolliti mälus oleva SQL andmebaasi vastu.

Rakenduse API testimiseks kasutati musta kasti põhimõttel Javascriptis kirjutatud teste, mida jooksutati Postman rakenduses. Nende testide eesmärk oli kontrollida, kas rakenduse API toimib nagu klient eeldaks, ning kas tagastatakse õiged vastused õigete koodidega.

Lisaks testiti kogu süsteemi ka genereerides MQTT vahendajasse hulga andmeid, mis jäljendaks seadmete reaalset kasutust, milleks kasutati vastavat vabavarana kättesaadavat rakendust IoT-Data-Simulator.

4.5.6 Andmesalvestuskihi analüüs

Arhitektuur *backend* veebirakenduse jaoks loodi meeskonnaprojekti käigus eesmärgiga, et see võimaldaks reeglite abil üsna lihtsalt luua uusi funktsionaalsusi ning laiendusi vastavalt nõuete muutumisele. Kuusnurkne arhitektuur võimaldas selge struktuuri alusel juurde lisada seadmete automaatse registreerimise funktsionaalsuse ilma muid rakenduse koodi osi muutmata.

Esimene versioon seadmete registreerimiseks koosnes ühest päringu aadressist, mis võttis vastu korraga objekti, mis sisaldaks nii seadme identifikaatorit kui ka seadmetelt tulenevaid andmeid. Kõik need salvestati sündmuste põhiselt ning siis hakkas rakendus otsima, kas uus sõnum sisaldas veel tundmata seadme identifikaatorit. Sellise lähenemise eeliseks oli kõikide sõnumite talletamine, mille põhjal oleks võimalik igal ajal rekonstrueerida sündmuste ajalugu ning uute seadmete avastamine. Üheltpoolt oli see kooskõlas ka Rahimi, Zibaeenejad & Safavi pakutud arhitektuuriga, kus kõigepealt toimub andmete salvestamine.

Siiski, kuna eesmärk oli talletada unikaalsete seadmete info andmebaasi ning seda kontrolliti iga telemeetria järel, tekkis olukord, kus iga uue sõnumi vastu võtmisel toimus ühe REST päringu raames mitu toimingut sest kõigepealt salvestati info ning seejärel otsiti, kas seade on juba olemas ning kui mitte, siis registreeriti kõigepealt äärepealne seade ning siis alamseade. See ei vasta REST stiili soovitustele, et iga liides peaks olema ühetaoline ehk teisisõnu iga ressurss peaks olema unikaalne [15]. Seega tuleb kliendile luua üheselt mõistetav ning kasutatav ressurss ehk päringuaadress. Seadmete registreerimine ning nende andmete edastamine on kaks erinevat tegevust erinevate nõuetega, mistõttu on nad tehtud eraldi ressurssideks.

Seega on rakenduse põhifunktsioonis kolm eraldi päringuvõimalust, mis teeb rakenduse kasutamise kliendile selgemaks ning võimaldab luua kliente. See võimaldab luua ka klienti, mis ei hakka edastama andmeid kui seadme registreerimine ebaõnnestub, aidates kaasa süsteemi töökindlusele ning jõudlusele kuna jäävad tegemata ebavajalikud päringud. Samas on kõikides meetodites struktuurne logimine, mis aitab leida vigu ning monitoorida rakenduse toimimist.

Selle lahenduse puhul on seadmete registreerimise tõeallikaks andmebaasi tabel Devices. Realiseeritud ei ole seadmete maha registreerimist või nende staatuse muutmist, kuna töö keskendus automaatse avastuse konfigureerimisele, kuid selle funktsionaalsuse lisamine on tehtud arhitektuursete otsuste ning disaini tõttu tehtud üsna lihtsaks.

Koodis peeti tähtsaks domeeni iseseisvust, testitavust ning vajadust lihtsaks laienduseks, kui ärilised nõuded kasvavad või muutuvad. Oluline nõue oli, et seadmeid saaks vajadusel siduda ettevõtte varadega. Kuusnurkne arhitektuur võimaldab *backendi* liidestada ettevõtte kolmandate süsteemidega luues vastavad adapterit, mis küsivad vajadusel infot. Laudade broneerimise süsteemi näitel oleks vajadus siduda laudade staatust näitavaid ekraane lauaga ning laua füüsilise asukohaga.

4.6 Haldusteenuste kiht

Haldusteenuste kiht (ingl. *management*) on mõeldud süsteemi liidestamiseks teiste süsteemidega. Siia alla võivad kuuluda teised rakendused, kust küsitakse andmeid või andmete edastamine teistele rakendustele.

Lõppkasutajaks süsteemis on IT seadmete või süsteemi haldur, kelle vajadus on hallata seadmeid kasutajaliideses. Selleks kasutatakse vabavaralist ThingsBoard platvormi, mille litsents lubab seda kasutada ka kaubanduslikel eesmärkidel. Täpsemalt on kasutusel ThingsBoard demo versioon, mis on veebipõhiselt kõigile kättesaadav [16].

ThingsBoard on valmislahendus, mis on mõeldud IoT-seadmete haldamiseks. Seda saab üles seada nii lokaalses serveris või klastris või saab kasutada nende pilveteenust. Vabavarana on nende koodirepositoorium samuti avalik ning seda on lubatud muuta. ThingsBoard võimaldab lisaks seadmete haldamisele luua seadmete andmete põhjal töölaudu (*dashboard*) ning teavitusi.

4.6.1 Liidestamine ThingsBoard lahendusega

ThingsBoardil on oma andmebaas ning veebiliides, mille kaudu saab teda kasutada programmi abil. Info seadmete kohta on talletatud eelpool kirjeldatud tagarakenduse kaudu kontrollitavas andmebaasis. Selleks, et kasutada ära ThingsBoard võimalusi, tuleb seadmed ka ThingsBoardis registreerida.

Seadme avastamise järgselt salvestatakse ta tagarakenduse poolt andmebaasi, ning kui see on edukas, registreeritakse sama seade ThingsBoardis REST päringuga. Kui see on edukas, siis salvestatakse seadmete tabelisse ka ThingsBoard ID, mis on vajalik seadmete andmete edastamiseks ThingsBoard kasutajaliidesesse.

4.6.2 Sündmustepõhine disain väliste päringute jaoks

Selleks, et vältida koodis klasside omavahelisi sõltuvusi ilma liidesteta, on kasutusel Spring Application Events. See täidab sama eesmärki, mida Gang of Four välja pakutud „mediaatori“ ehk vahendaja muster [17]. Iga *event* ehk sündmuse jaoks on oma klass ning seal kus vaja, luuakse koodis selle sündmuse objekt – sündmus on koodi mõistes fakt, et mingi toiming on läbi viidud. Seejärel lisatakse see sündmus Spring Events haldajasse, mis käitub kui vahendaja. Klass, kus sündmus luuakse, ei pea teadma ega sõltuma klassist, kus sündmuse järel toiminguid tehakse ning vastupidi.

Pärast seadme salvestamist andmebaasi luuakse sündmuse objekt (EdgeSaved) ning saadetakse ta vahendajale. Backendi teenuste kihis on sõltumatu teenus RegisterWithThingsBoard, mis kuulab kõiki EdgeSaved ja LeafSaved sündmusi vastavates meetodites. Sündmuse toimumise järel tehakse teenusest liidese kaudu päring.

Kuna liides peidab adapteri tegelikku rakendamisviisi, teab ainult adapter ThingsBoardist ning teab kuhu teha päring.

Kui Thingsboardi APIst saadakse vastu Ok 200, siis kasutab rakendus sarnast sündmuste vahendamise loogikat selleks, et uuendada seadme infot andmebaasis ning salvestada lisaks ThingsBoardi ID mis on GUID vormis.

Pärast seadmete registreerimist on võimalik saata telemeetria päring, mis kasutab sarnast koodi ülesehitust selleks, et kõigepealt seadmete info salvestada ning seejärel Spring Events abil teha päring ThingsBoardi, millel antakse kaasa ka seadme ThingsBoardi identifikaator.

4.6.3 Manageerimise kihi analüüs

See kiht vastutab tagarakenduse suhtluse eest kolmandate süsteemidega, milleks on lahenduses kasutatud ThingsBoard platvormi ning Azure Active Directory kataloogiteenust. Tagarakenduse kuusnurkne arhitektuur ning sündmustepõhine disain võimaldab uusi funktsionaalsusi luua tänu selgele struktuurile lihtsamalt – sealhulgas liidestamist ettevõtte süsteemidega.

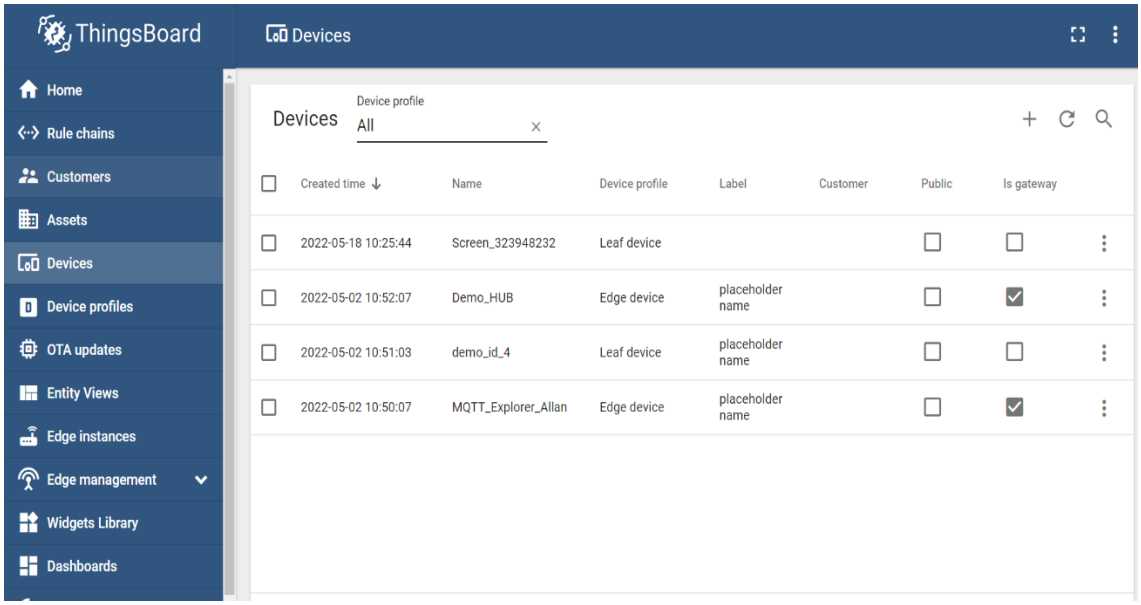
Selleks, et edastada infot seadmetele, on tagarakendus ühendatud otse MQTT vahendajaga. See on meeskonnatöö projektis loodud lahendus, kuid ei vasta seadmete avastamise jaoks loodud arhitektuurile. Tagarakendus peaks suhtlema ainult REST alusel ning ei tohiks ise olla kliendiks MQTT vahendajale – teisisõnu ei peaks tagarakendus sõltuma teistest standarditest. Seega tuleb sõltuvused ka siin ümber pöörata ning luua sarnaselt avastamise rakendusele oma rakendus, mille eesmärgiks on sõnumite tõlkimine ning edastamine – ehk sõnumite edastamisel MQTT vahendajasse.

Laudade broneerimise süsteemi terviklahenduse puhul on kasutusel ka teise tudengitiimi poolt loodud võimekus kasutada rakenduse andmebaasi äriteabe analüüsi jaoks. Selles lahenduses pärib *Power Bi* serverirakendus andmeid lahenduse andmebaasist. Seega oleks võimalik seda võimekust kasutada seadmetega seotud ärianalüüsiks.

4.7 Rakenduse kiht - kasutajaliides

Lahenduse kasutajaliidesena on kasutusel ThingsBoard Community Edition, mis on vabavaraline platvorm IoT-seadmete andmete kogumiseks. ThingsBoard UI on loodud

Angular 9 raamistikuga, mis põhineb komponentide põhisel lähenemisel. Seega on võimalik muuta rakenduse stiilielemente vastavalt ettevõtte vajadustele ühildada rakendust enda stiilinõuetega. ThingsBoard UI seadete vaade sisaldab halduseks vajalikke võimalusi, sealhulgas andmeväljade põhjal sorteerimist ning filtreerimise ja otsingufunktsioone (joonis 20).



Joonis 20. ThingsBoard seadete vaade.

4.7.1 ThingsBoard UI arhitektuur

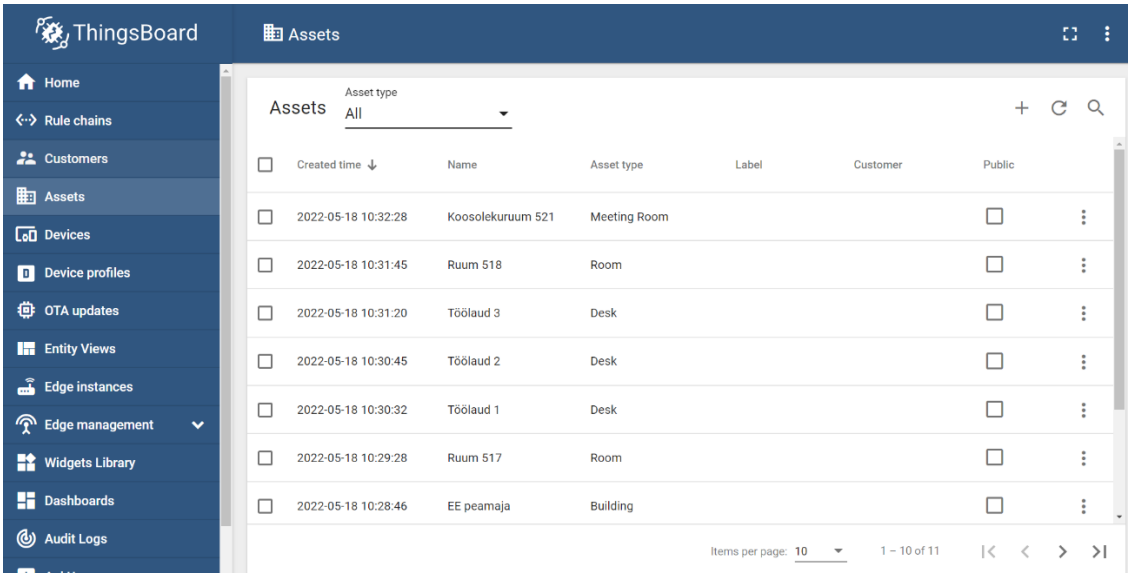
ThingsBoard on samas platvorm, mis tähendab, et tal on olemas enda tagarakendus ning andmebaasid, samuti pakutakse seda SaaS pilveteenusena. ThingsBoard Core ehk tagarakendus on kasutatav REST API kaudu, mida kasutab ka ThingsBoard UI.

Seadmete andmete tõeallikaks on autorite loodud tagarakenduse valdkonnamudel ning selle andmebaas. Selleks, et andmed jõuaks ka kasutajaliidesesse, saadetakse andmed ThingsBoard REST liidese kaudu ThingsBoard tagarakendusse. ThingsBoardi andmete struktuur on piisavalt üldine, et tagada nõuete täitmine.

4.7.2 Varadega sidumine

ThingsBoard võimaldab ka varasid luua ning neid seadmetega siduda. Joonisel 21 on näide kasutusjuhust, kus varadena on kujutatud kontor, ruumid ning lauad. Üheltpoolt tuli arvestada ettevõtte poolt seatud nõudega varade ja seadmete sidumise kohta ning teisalt seda, et paljudel ettevõtetel on oma varahaldussüsteemid juba tõenäoliselt olemas.

Juhul, kui ettevõttel on varahaldussüsteem olemas, või päritakse need näiteks kataloogteenusest, võimaldab haldusteenuste kihi tagarakendus neid liidestada ja ThingsBoardis ka varad REST liidese kaudu registreerida. Teiseltpoolt on võimalus, luua varade haldus otse ThingsBoardis. Mõlemil juhul toimuks hetkel varade ja seadmete omavaheline sidumine kasutaja poolt ThingsBoard UI kaudu, mis salvestab need seosed ThingsBoardi andmebaasi.

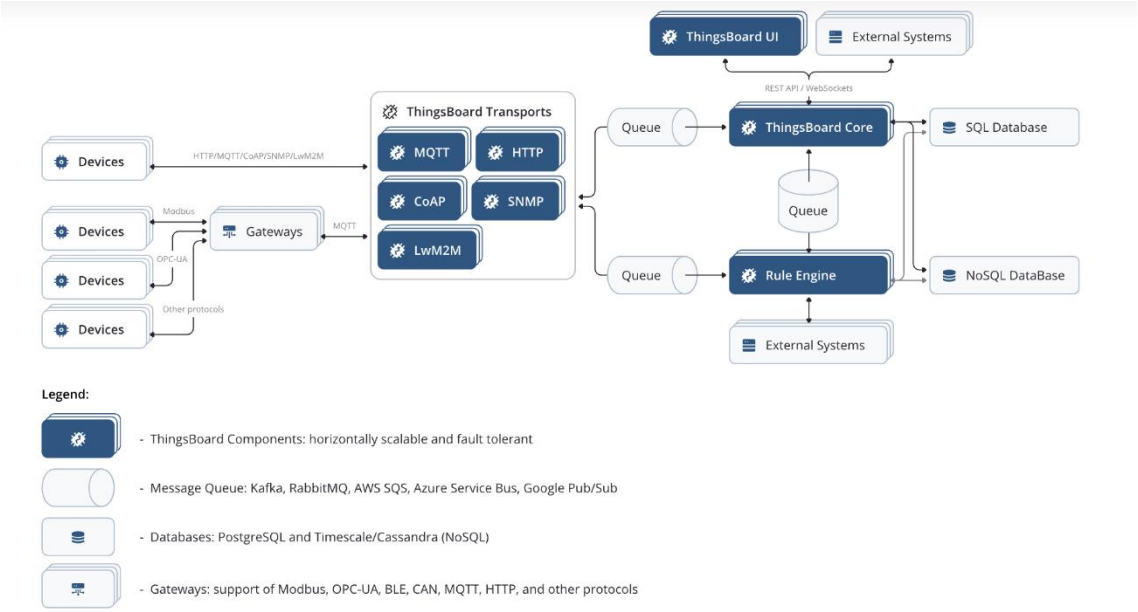


Joonis 21. ThingsBoard varade vaate võimalus.

4.7.3 Rakenduse kihi analüüs

Oluline nõue oli tagada liidestamise võimalus ettevõtte süsteemidega nagu varahaldustarkvara või mõne veebipõhise kataloogteenusega, mis on ettevõtetes laialt kasutusel. Üks võimalus oleks võtta ThingsBoard kood ning seda vastavalt vajadusele muuta, kuna litsents seda võimaldab. Ilma ThingsBoard koodi muutmata ei ole aga võimalik, et ThingsBoardi enda tagarakendus küsiks andmeid kolmandatest süsteemidest.

Kuna valmislahenduse koodiga tutvumine ning selle muutmine on töömahukas protsess ning samuti tuleb arvestada võimalike turvakaalutlustega ettevõtte süsteemidega liidestamisel, otsustasid autorid ThingsBoard lahenduse esialgu võtta kasutusele selliselt, et autorite pakutud lahenduse tagarakendus on seadmete info tõeallikaks ning samuti rakenduseks, mille kaudu liidestada kolmandaid rakendusi.



Joonis 22. ThingsBoard Community Edition arhitektuur. Allikas: [18].

Joonisel 22 on näha ThingsBoard platvormi üldine arhitektuur. Autorid on kasutanud ainult ThingsBoard Core, Rule Engine ning ThingsBoard UI võimalusi. ThingsBoard võimaldab seadmetelt saadud andmete põhjal (*telemetry*) luua aegridu ning teavitusi. Kasutaja saab vastavalt vajadustele seadistada nii töölaudu kui Rule Engine kaudu teavitusi. Need on nõudeid täitvad funktsionaalsused, mille arendamiselt on seetõttu võimalik tööressursse kokku hoida.

Lõpuprojektis välja pakutud lahenduses toimub nii seadmete registreerimine ThingsBoardi kui ka tema andmete edastamine ThingsBoard Core REST kaudu, mida vahendab tagarakendus. ThingsBoard võimaldab seadmeid kuulata ka iseseisvalt kahel järgneval viisil.

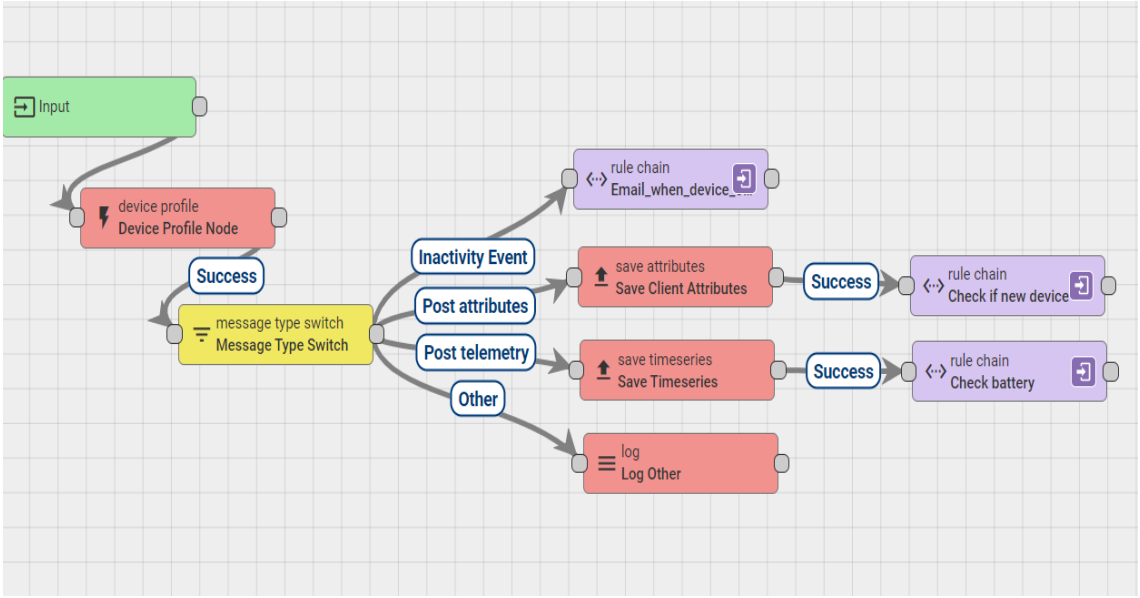
Esiteks oleks võimalus, et seadmed saadavad andmeid ThingsBoard Transportidele, kuid sellisel juhul tuleb need seadmed enne ThingsBoardis siiski kas programmaatiliselt või käsitsi registreerida, misjärel luuakse seadmele võti. MQTT protokolli näitel oleks seade ise MQTT kliendiks ning ta peab ThingsBoardi MQTT vahendajaga ühenduma kasutades seadme registreerimisel saadud võtit/võtmeid. Kui REST kaudu ThingsBoard Core'i seadmeid luua, on võimalik ise oma võtmed kaasa anda. ThingsBoard Transportide kasutamise eeliseks on võimalus edastada seadmete andmeid vabal kujul – teisisõnu ei pea ette määrama seadme poolt edastavate andmete struktuuri, et saaks neist omakorda luua töölaudu ning seadistada teavitusi.

Teine võimalus oleks eraldi tööle rakendada ThingsBoard IoT Gateway, mis on mõeldud eelkõike legacy ning kolmandate süsteemidega ühendamiseks. ThingsBoard IoT Gateway saab seadistada konfiguratsioonifailidega, kuid sellisel juhul peab ette määrama nii seadmete info kui ka nende poolt edastavate andmete struktuuri. Selle Gateway puhul oleks võimalus kasutada automaatse avastuse võimalusi ning sarnaselt avastusrakendusele luua iga äärepealse seadme kohta uus rakendus. Selle lähenemise suurimaks puuduseks on vajadus ette genereerida seadmete andmete kuju.

4.8 Koostöö ja protsesside kiht

Eelnevate kihtide info pole kasulik, kui see ei käivita tegevust [6]. Tegevused käivitatakse reeglistike alusel. Lahenduse funktsionaalsusetes nõuetes on kirjas, et haldur peab saama luua reeglistikke automaatteavitusteks, lisaks peab haldur saama teavituse kui uus seade on süsteemi registreeritud või kui seade pole ammu võrgus viibinud.

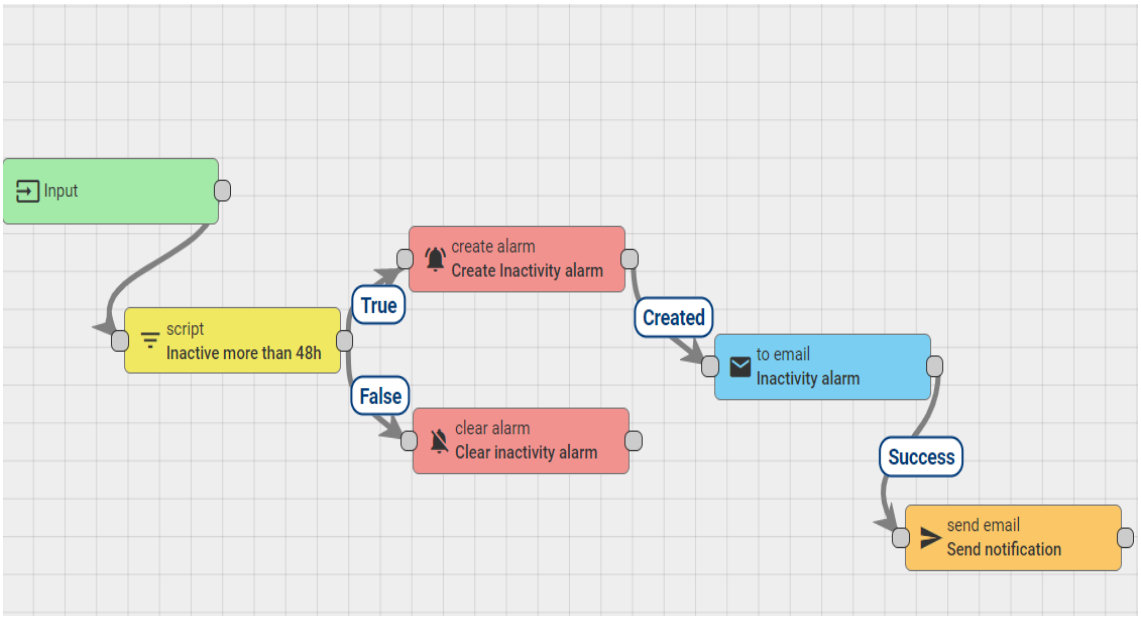
Neid nõudeid täidab ThingsBoard Rule Engine, mis võimaldab reegleid seadistada visuaalse kasutajaliidese kaudu. Kasutaja saab ettemääratud sündmuste juhtumise põhjal visuaalselt tegevusi üksteise järel programmeerida. Iga Thingsboardi saabunud sõnum läbib baasreeglistikku (joonis 23), kuhu lisatakse enda loodud reeglid. Võimalikud sisendid, mille põhjal on võimalik reegleid seadistada on näiteks seadmete elutsükli sündmused või nende poolt edastatud telemeetria. Igat reeglit on võimalik täpsustada kirjutades selle jaoks JavaScriptis meetodi, mis salvestatakse ThingsBoard andmebaasi.



Joonis 23. Lahenduse baasreeglistik ja sinna külge lisatud reeglistikud.

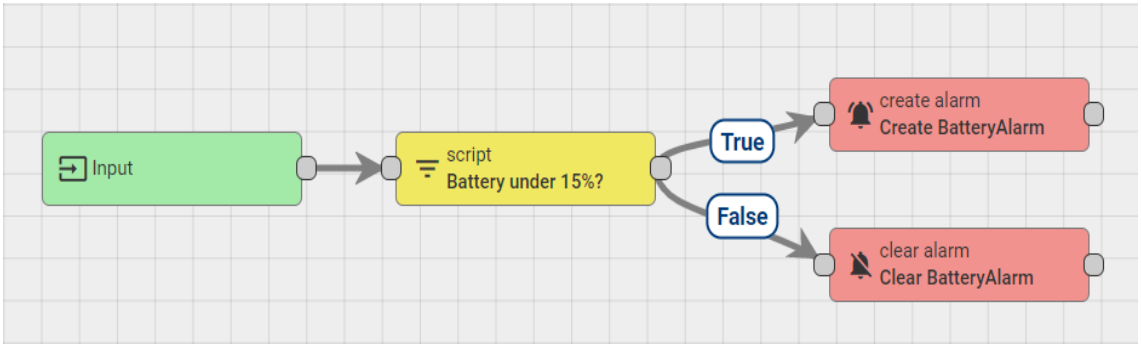
Näitena on lahenduses seadistatud kasutaja teavitamine järgmistel juhtudel:

1) e-maili teel seadmest, mis on olnud võrgust väljas kauem kui 48h (joonis 24)



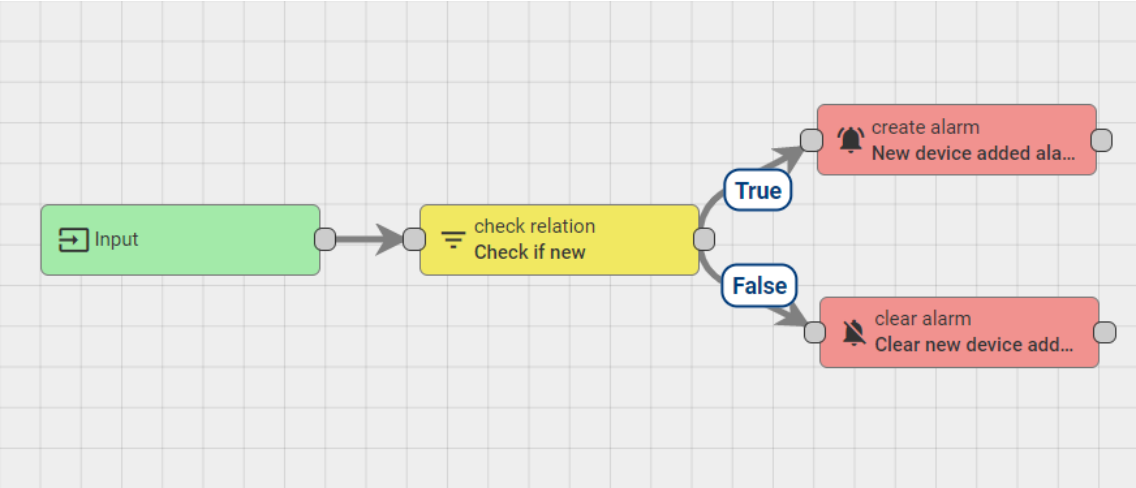
Joonis 24. Reeglistik kasutaja teavitamiseks, kui seade on olnud 48h võrgust väljas

2) teavitusfunktsioon töölaual seadme kohta, mille patarei vajab vahetamist, näiteks patarei on alla 15% (joonis 25)



Joonis 25. Reeglistik teavituse saatmiseks kui patarei on liiga madal

3) töölaua teavitus uue seadme registreerimisest (joonis 26)



Joonis 26: Reeglistik uue seadme registreerimisel teavituse saatmiseks

5 Lahenduse analüüs ja hinnang

Selles peatükis analüüsitakse lahendust tervikuna ning antakse hinnang protsessile. Analüüsitakse sätestatud nõuete täitmist, lahenduse tugevusi ja nõrkuseid ning tuuakse välja võimalikud edasiarendused. Lisaks näidatakse, millised olid agiilses arenduses suurimad pöördepunktid.

5.1 Üldine hinnang

Autorite hinnangul täidab pakutud lahendus eesmärki lihtsustada ettevõttes suure hulga IoT-seadmete ja nende alamseadmete süsteemis registreerimist ja haldamist. Sellise lahenduseni jõuti kaaludes palju võimalikke lahendusi nõudmiste vastu. Kuigi üldiselt suutsid autorid eesmärki täita, on töös veel palju parendamisvõimalusi ning vajadus arendusteks, et kõiki nõudeid täita.

Tööd tehes on lähtutud on eeldustest, et täiuslikku lahendust pole – autorite töö on üks paljudest võimalustest eesmärki täita. Kuna rõhku on pandud vabavaralistele komponentidele ning hajusale süsteemile, siis see loob eeldused, et välja pakutud lahendus võib olla mõne teise lahenduse algpunktiks kuna komponente on võimalik välja vahetada.

5.2 Agiilse arenduse protsess

Agiilse arenduse alustalaks on pidev lahenduse ja plaani ülevaatamine ning vajadusel muudatuste sisse viimine võimalikult kiirelt. Agiilse arenduse üks olulisi soovitusi on teha üks toimiv tükk valmis ning see võimalikult ruttu tagasisidestamiseks kliendile viia [4]. Valida tuleb kõige suurema väärtusega tükk kõigepealt. Lähtuvalt nendest põhimõtetest, toimus arenduse käigus seetõttu mitu olulist kannapööret ehk suuremat suunamuutust.

Esialgu oli plaanis arendada seadmete ning varade haldusportaal ise. Sarnaste lahenduste uurimisel selgus, et on olemas vabavaraline lahendus ThingsBoard, mis katab peaaegu kõik haldusportaalile seatud funktsionaalsed nõuded. Autorid leidsid, et pole mõistlik sama funktsionaalsusega portaali hakata ise arendama, kuid vajas aega, et selgitada välja võimalikud viisid, kuidas ThingsBoard platvormi on võimalik ning mõistlik kasutada.

Teiseks kannapöördeks oli Akri, avastusskripti ja avastusrakenduse toomine ühte klastrisse. Esialgne plaan nägi ette, et automaatne avastus toimub Eesti Energia enda klastris, kust saadetakse avastatud seadme info autorite klastris olevasse tagarakendusse. Selle lahendusega ilmnes kaks probleemi. Esiteks, kuna avastusskript vajas muutmist autorite poolt, tähendas see, et Eesti Energia oleks pidanud selle jaoks looma uue klastri ressursi. Sellega kaasneks muudatuste juhtimise ahel iga kord, kui on vaja teha koodi muudatusi Eesti Energia klastris jooksvates konteinerites. Teiseks ning olulisemaks põhjuseks oli, et ise automaatavastuse klastris üles seadmine annab hea ülevaate süsteemi toimimisest ning pakkus hea võimaluse uute teadmiste omandamiseks ning tervikliku lahenduse loomiseks.

Kolmandana tuli teha palju otsuseid arendusplaanis olevate funktsionaalsuste olulisuse järjekorra kohta, mis tähendas, et lõpuprojektist jäi välja osade nõuete täielik täitmine. Näiteks tuli tahaplaanile jätta varade automatiseeritud lisamine haldusportaali, mille asemel seadsid autorid esikohale automaatavastusega ning seadmetega seotud funktsionaalsuseid. Projekti alguses plaane tehes alahindasid autorid osade tehnoloogiatega tutvumisele ning nendega komponentide seadistamisele kuluvat aega. Kui sai selgeks, et kõike teha ei jõua, siis lähtuti agiilsetest väärtustest, mille abil valiti välja olulisemad funktsioonid, mis sooviti kindlasti enne arenduse lõppu realiseerida. Toote *backlogis* märgiti ülejäänud ülesanded sildiga „tuleviku idee“, et plaanidest jääks

märk maha ning kui projekt peaks jätkuma oleks võimalik seda elihtsamini edasi arendada.

5.3 Lahenduse vastavus nõuetele

Lahendus täidab suures osas ärilisi eesmärke: registreerib ühendunud seadmed automaatselt haldusportaali ja võimaldab seadmete sidumist varadega. Lisaks on võimaldatud reeglistike koostamine, mille alusel saata teavitusi ning seadmelt saabunud telemeetriast saab luua visuaalseid representatsioone.

Siiski olid mõned funktsionaalsed nõuded, mis jäid täitmata. Üheks nõudeks oli, et varadele ning seadmetele saab luua kategooriaid, mille alusel saaks neid hiljem rühmitada. ThingsBoard lahenduse valimisel jäi esialgu mulje, et antud see funktsionaalsus on platvormil võimaldatud. Hiljem selgus aga, et see funktsionaalsus on ainult ärikasutaja versioonil ning puudub vabavaralisel tarkvaral. Kuna tegu pole ärikriitilise nõudega, otsustati see hetkel tahaplaanile jätta ning funktsionaalsuse lisamine vabavara koodile lisati tulevikuplaanidesse.

Teiseks nõudeks, mis jäi realiseerimata oli varade automatiseeritud toomine haldusportaali. Selle nõude täitmiseks oli plaan kasutada ThingsBoard API-t. Plaanis oli luua liides tagarakendusse, mis küsiks ettevõtte varade halduseks kasutavast süsteemist (Azure Active Directory), varad ning tagarakenduses asuv liides edastaks need ThingsBoard API kaudu haldusportaali. Autorid prioritiseerisid seadmete automaatset registreerimist ning seadmete liideste arendamist. Samas on võimalik ThingsBoard haldusportaalis varasid luua kasutajaliideses käsitsi, millest piisas lahenduse sobivuse testimiseks.

Kolmandaks täitmata nõudeks on, et haldur saab vaadata seadmeid, mis on varaga seotud või sidumata. Seda nõuet on võimalik ThingsBoardis rakendada, kuid selle jaoks tuleb luua uus vaade, mis on seotud reeglistikuga, mis kontrollib, kas seadmele on määratud suhe mõne varaga. Autoritel ei jäänud piisavalt aega, et funktsionaalsust ThingsBoardis võimaldada.

Neljandaks jäi osaliselt rakendamata seadete haldamise puhul nende süsteemist kustutamine. Kasutaja saab küll ThingsBoard rakenduses seadistada teavituse selle kohta, kui mõnda seadet pole kasutaja poolt määratud aja jooksul enam nähtud – teisisõnu kui

ta pole mingi perioodi jooksul andmeid edastanud. Samas ei saa kasutaja hetkel ThingsBoard UI kaudu seadmeid tagarakenduse süsteemist eemaldada. Autorite hinnangul ei olnud see kasutuslugu projekti peamine fookus, kuid süsteemi tegelikuks toimimiseks oleks see arvatavasti vajalik. Samas võib ettevõte otsustada, et infosüsteemist pole seadeid vaja eemaldada ning võib süsteemi kasutada selliselt, et luua eraldi vaade mitteaktiivsetest seadetest.

Mittefunktsionaalsed nõuded said kõik täidetud, välja arvatud info terviklikkuse nõue. Kuna lahenduses on hetkel kasutusel kaks andmebaasi, tagarakenduse ja ThingsBoard andmebaas, on seadmete info osaliselt dubleeritud. Terviklikkuse tagamiseks on vaja, et üks andmebaas oleks tõe aluseks. Selle jaoks oleks vaja süsteem viia ainult ühele andmebaasile, nt kasutada ThingsBoard UI jaoks loodud tagarakendust ja selle andmebaasi või ellu viia andmete tervikluse kontrollid andmebaaside vahel. Näiteks kui seade kustutatakse ThingsBoard rakenduses, peaks see kustuma ka lahenduse enda andmebaasis.

Samas tuleks kaaluda, kas andmete kustutamine on vajalik, või pigem eelistada seadmete staatuse klassifikaatori loomist. Kuna töös oli esimene rõhk seadmete avastamisel ning võimaldada teavitada kasutajat seadmete võrgus mitteaktiivseks muutumisest, siis leidsid autorid, et esialgu piisab ka sellest, et tekibki automaatselt nimekiri mitteaktiivsetest seadmetest ning sealt andmete kustutamine ei pruugigi olla esialgu vajalik.

5.4 Lahenduse tugevused

Autorite hinnangul oli lahenduse üks huvitavamaid osasid arhitektuuri loomine ning erinevatest olemasolevatest lahendustest terviku lahenduse kokkupanek. See nõudis muudatusi ning liideste programmeerimist. Alljärgnevalt toovad autorid endi hinnangul kõige olulisemad tugevused, millega autorid kõige rohkem oma töös rahul on.

5.4.1 Tehnoloogiate kombineerimine

Kombineerides nii tuntud tehnoloogiaid kui uusi lähenemisi nagu Akri, suutsid autorid luua lahenduse, mis võimaldab ettevõttel üsna lihtsalt seadistada IoT-seadmeid. Kui süsteem on seadistatud, siis pärast võrku ühendamist, salvestatakse seadmed andmebaasis ning nende poolt edastatavate andmeid on võimalik kasutajaliideses hakata kasutama suhteliselt väikese vaevaga.

5.4.2 Laiendatav

Autorite hinnangul on üheks olulisemaks tugevuseks lahenduse juures tema laiendatavus. Laiendatavust peeti silmas nii eritüüpi seadmete puhul, kui ka tagarakenduses ärireeglite ning teiste programmide liidestamisel. Näiteks on võimalik lisada Akrile mitu erinevat avastusskripti, et võimalik oleks ära kasutada mitmeid erineva suhtlusprotokolliga seadmeid. Lisaks on tagarakendus üles ehitatud selliselt, et adapterite abil saab sinna lisada erinevaid andmebaase ning liidestada seda väliste programmidega, ilma suuri koodimuudatusi tegemata.

5.4.3 Paindlik juurutamine

Kuna kogu rakendus on Kubernetese abil pilveteenuses jooksutatav, saab ära kasutada dünaamilist jõudluse suurendamist, juhul kui lahendus vajab IoT-seadmete süsteemi suurenedes rohkem jõudlust ning mälu.

Rakenduse konteineriseeritus ja valmidus Kuberneteses orkestreerida muudab lahenduse ülesseadmise väga kiireks ja lihtsaks. Vaja on ainult Kubernetese keskkonda seadmes või pilvepakkujas ning jooksutada ülesseadmise käsud. Azure Kuberneteses Service platvormil rakenduse jooksutamiseks on loodud ka Github Actions käsud, kus tuleb asendada saladustena hoitud klastri kredentsiaalid ning rakendus seatakse platvormil üles automaatselt. Vajalik on ainult juhtseadmete suhtluse seadistamine MQTT vahendajaga, mida tuleks teha ka seadmete tavalisel ülesseadmisel.

5.4.4 Vabavara kasutamine

Vabavara kasutus lahenduse moodustamiseks pakub samuti mitmeid eeliseid. Esiteks ei ole lahenduse kasutamine sõltuv ühestki teenusepakkujast. Hetkel kasutusel oleva Azure keskkonna võib vabalt asendada mõne teise pilveteenuse pakkujaga või jooksutada kogu koodi täiesti oma serveris. See annab paindlikkuse firmale valida endale sobivaim variant ning saab teenusepakkujat vabalt vahetada, ilma et peaks hakkama arendama uut süsteemi. Teiseks saab vabavara lähtekoodi alla laadida, mis pakub võimalusi mugandada rakendusi enda vajadustele. Lahenduses valitud vabavaralised projektid omavad ka aktiivset arendajate kogukonda, mis tähendab pidevat edasiarendust ning tuge küsimuste korral.

5.4.5 Lihtne seadistamine

Lihtsustatud on ka rakenduste konfigureerimine. Selleks on loodud eraldi konfigureerimise failid (joonis 27), mis sisaldavad endas näiteks keskkonna muutujaid ning ühenduste loomiseks vajalikke aadresse. Seega pole konfigureerimiseks vajalik rakenduse koodi muutmine, piisab ainult väärtuste muutmisest konfiguratsiooni failides.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-conf
data:
  default.conf: |
    server {
      listen 80;
      listen  [::]:80;
      server_name  localhost;
      location / {
        root   /usr/share/nginx/html;
        index  index.html index.htm;
        try_files $uri $uri/ /index.html;
      }
      location /assets/desks {
         proxy_pass http://desk-booking-api:8080;
      }
    }
```

Joonis 27. Configmap konfiguratsioon.

5.5 Lahenduse puudused

Hajusad süsteemid on loomupoolest keerulised, kuna neid on raskem testida, sest nad sõltuvad andmete vahetamisest üle võrgu. See kehtib ka autorite töö puhul. Autorid on teadlikud, et pakutud lahendus pole ideaalne ning igas olukorras sobiv, vaid pigem tuleks seda vaadelda kui alguspunkti, mis võimaldab IoT-seadmete süsteemi üles seada. Suur faktor lahenduse nõrkuste juures on seotud projekti ajaliste piirangutega. Allpool on toodud välja autorite silmis nende lahenduse suurimad puudused.

5.5.1 Optimeerimine

Puudulikuks jäi lahenduse optimeerimine, mida vajaksid eelkõige avastusrakendus ning tagarakendus. Seda nii toimimiskiiruse vaatenurgast, näiteks asünkroonse mitmelõimelisuse implementeerimine mõlemasse rakendusse ning ka jõudluse ja mälu nõudluse seisukohast.

5.5.2 Andmete jäik struktuur

Hetkel suurimaks probleemiks on avastusrakenduse sõltumine kasutusolevate seadmete sõnumi struktuurist. Uut seadet, mis edastab teisel kujul telemeetriat, hetkel süsteemis registreerida võimalik ei ole. Selle lahendamiseks on autoritel plaan tulevikus kasutada JSON formaadis sõnumi salvestamine andmebaasi, mis võimaldaks tagarakendusel vastu võtta ning andmebaasi salvestada ükskõik mis kujul andmeid. Selle rakendamisel tuleb aga hoolikas olla ning seejuures kaaluda nii turvalisuse küsimusi kui ka seda, kas selle jaoks on mõistlik relatsioonilise või mitterelatsioonilise andmebaasi kasutamine.

5.5.3 Vabavara turvalisus

Vabavara kasutuselevõtuga kaasneb samuti mõningaid puuduseid. Näiteks tuleks enne lahenduse ettevõtte siseselt kasutusele võtmist kindlasti auditeerida vabavaraliste lahenduste kood, et tagada turvalisus ja vältida vabavara kaudu ettevõtte süsteemidesse pääs.

5.5.4 Teadmised pilvetehnoloogiatest

Autorite pakutud lahendus eeldab ettevõttes häid teadmisi pilvetehnoloogiatest ning nende optimaalsest kasutamisest. See oli valdkond, mis autorite endi jaoks nõudis palju uute teadmiste omandamist.

5.6 Võimalused edasiarendusteks

Kuna mõningad funktsionaalsed nõuded jäid täitmata, tuleks prioritiseerida nende arendust. Selleks, et süsteem vastaks kõigile seatud nõutele on vajalik arendada kategoriseerimise funktsionaalsus, sidumata ja seotud seadmete vaatamise funktsionaalsus. Lisaks on vajalik luua liides varade automatiseeritud toomiseks ThingsBoard portaali. Vajalik on muuta avastuse rakendust selliselt, et see suudaks vastu

võtta igal kujul sõnumeid, et tagada laiendatavus. Tegeleda tuleb ka terviklikkuse tagamisega. Veel on vajalik rakenduste jõudluse ja mäluvajaduste optimeerimine.

Üheks kasulikuks edasiarenduseks oleks ThingsBoardi rakenduses vaikimisi vaadete ning reeglistike loomine. Hetkel on loodud ainult hädavajalikud või võimalusi näitlikustavad vaated ja reeglid. Ideaalis võiks olla loodud piisavalt lahendust toetavaid erinevaid vaateid, töölaua põhjasid ning reegleid, et lahenduse haldur lihtsalt valib, milliseid ta soovib kasutada. See muudaks lahenduse halduse poole implementeerimise oluliselt kiiremaks ning tagaks, et haldur ei pea ise oskama ThingsBoardis luua reegleid ning vaateid.

Autoritel oli algselt plaan testida lahendust ka teistsuguste IoT-seadmetega. Näiteks kasutada Zigbee ja MQTT vahelist tõlkeadapteri seadet ja testida, kas süsteem saab nende seadmete registreerimise ja info edastamisega hakkama. Kahjuks osutus uute seadmete ja adapterseadme kasutamine ja ülesseadmine liiga ajamahukaks ning selle asemel seati esikohale teiste lahenduste osade arendamist. Selle asemel kasutati aga seadmete simuleerimist saates automaatseid sõnumeid erinevatele teemadele MQTT vahendajas.

Tulevikus tuleks kindlasti proovida süsteemi kasutada mitmete eri tüüpi seadmetega. Selleks, et lahendust saaks kasutada võimalikult paljude eri protokolli seadmetega, tuleks kasutusele võtta ka teised Akri poolt toetatavad avastusskriptid. Lisaks on võimalus luua ise uusi avastusskripte, näiteks avastusskript Bluetooth seadmete avastamiseks.

Hetkel on süsteem orienteeritud pigem telemeetria vastuvõtmisele ning info edastamine juhtseadetele tehakse tagarakenduses. Edasiarendusena võiks avastusrakenduse muuta selliseks, et kuulamise asemel luuakse kohe automaatselt kahepoolne suhtluskanal, mille kaudu saaks juhtseadmele saata käsklusi ning vajalikku infot.

5.7 Projekti elluviimise hinnangteostuse tähelepanekud meeskonnatöö osas

Projekti viidi ellu põhimõttel, et autorid jagasid ülesanded omavahel ära ning vajadusel küsiti nõu ning aidati üksteist lahenduste leidmisel. Samuti arutati kõik ideed iganädalaselt läbi ning hallati projekti *backlogi* ja sealolevaid pileteid koos. Kuna töökohaks oli Eesti Energia kontor, tegi see suhtlemise oluliselt lihtsamaks ning tagas sujuva võimaluse üksteist nõustada ning aidata.

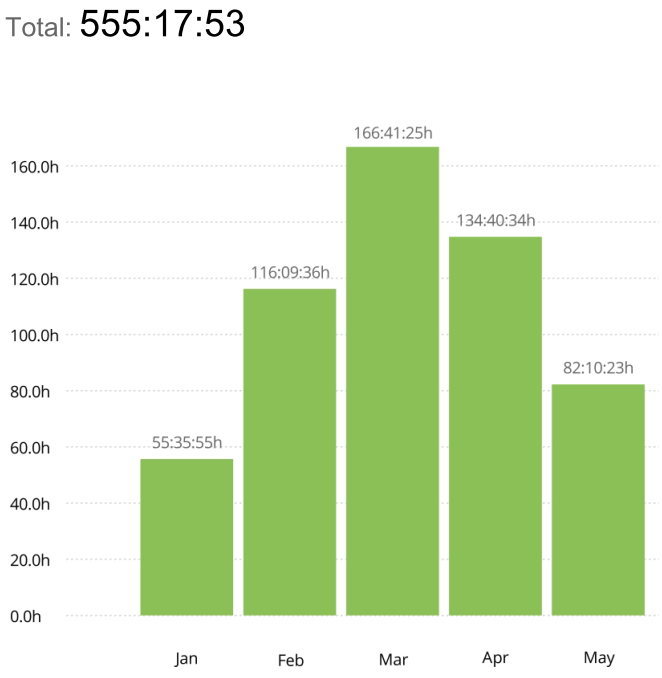
Semestri jooksul saadi jooksvat tagasisidet ettevõtte mentoritelt ning esitleti neile ka funktsionaalsuste demosid. Samuti kohtuti vajadusel juhendajaga, kes andis kasulikku tagasisidet ning juhiseid projekti eesmärkide seadmiseks ning tegevuste kirjeldamiseks.

Suurimaks takistuseks osutus raskus täpse hinnangu andmisel, et kui palju aega läheb projekti kõikide nõuete täitmiseks. Samas ei tekkinud konflikte selles osas, kui oli vaja agiilse planeerimise käigus otsustada prioriteetide üle – alati jõuti kokkuleppele.

Üleüldine hinnang projekti elluviimisele on autoritel hea. Eriti tuleb välja tuua, kui oluline on selles olnud ettevõtte toetav roll ning usaldus üliõpilaste vastu. Samuti oli projekti õnnestumisel oluline roll terve meeskonna heas koostöös, kuhu kuulusid lisaks autoritele veel neli tudengit.

5.8 Teostatud tööde logid ja meeskondlik hinnang

Meeskonna liikmed hindavad üksteise panust võrdselt. Projektile pühendatud aeg logiti rakenduses Clockify, mille alusel logiti kokku kahe autori peale üle 555 tunni nagu näha allpool (joonis 28).



Joonis 28. Clockify logitud töötunnid seisuga 18.05.

5.8.1 Ajalogide sisuline kokkuvõte nädala kaupa

Tabel 1. Allan Pälli logid.

Nädal	**Tegevused**
17.01	• Avastusrakenduse baasi loomine • Avastusrakenduse Dockeri pilt ja Ci/CD protsess käima • CI/CD ülesseadmine rakenduste docker hubi üleslaadimiseks
24.01	• Kubernetese juurutamise aitamine, klastri seadistuste kohta uurimine • Azure autentimise erinevate flowde kohta lugemine, testimine • Graph API testimine veebis ja Postmaniga, lugemine
31.01	• Lõpuprojekti backlogi planeerimine, täiustamine • Kubernetese seadistamine automaatseks juurutuseks Azure-i läbi GithHub Actionsite • Avastusrakenduse ühendamine MQTT vahendajaga
07.02	• Avastusrakenduse ühendamine MQTT vahendajaga • Avastusrakenduse andmete tõlkimise muudatused, testimine, refaktoorimine • Tagarakenduse automaatne autentimine MSAL4J teegiga Azure Active Directorys
14.02	• Parandused logide salvestamisel väljapoole konteinerit • Event sourcing lugemine ning tagarakenduses seadmete salvestamise ettevalmistus • Projekti backlogi täiendamine • Bugi lahendamine, millal panna kaasa token frontendist
21.02	• Projekti backlogi täielik ülevaatus, planning poker Ürgeniga • Docker build rakenduste parandused Java artifaktide osas, et toimiks nii kohalikult kui ka pilves ühtemoodi • Teiste meeskondade aitamine front end bugiga, mis takistasid edit/delete nuppude tööd
28.02	• Tagarakenduse seadmete salvestamise kõik kihidide kihtideDocker build rakenduste parandused Java artifaktide osas, et toimiks nii kohalikult kui ka pilves ühtemoodi

	• Tagarakenduse sündmustepõhise loogika katsetamine ning näite loomine • Azure klastris rakenduste käivitamine ja selle testimine koos Ürgeniga
07.03	• Andmete genereerimine IoT-seadmete simuleerimiseks • Simulaatori seadistamine ja katsetamine, simulaatori abil rakenduste ja flow testimine
14.03	• Github Actions kaudu juurutamine Kubernetesesse koos saladuste korraliku hoidmisega
21.03	• Äripoolele demo jaoks front end ja reserveeringute info saatmise debugimine ja lahendamine – uue *flowg*a Outlook vs UI ei toiminud reserveeringute edastamine korrektselt • Merge konfliktide lahendamine ja nendest tulenevate bugide lahendamine enne ja pärast demo
28.03	• ThingsBoard dokumentatsiooni lugemine, kohalikult Dockeris jooksutamine – probleemid õige image leidmise ja korraliku käivitamisega • Kontoris teiste IoT-seadmete katsetamine Home Assistanti ja enne Elabis kasutatud gatewayga • Andmete genereerimine IoT-seadmete simuleerimiseks ja simulaatori seadistamine ja katsetamine
04.04	• Tagarakenduse refaktoorimine, adapterite ja kontrollerite testimine • ThingsBoard API katsetamine Postmanis ja flow paika panek
11.04	• Avastamise rakenduse ja tagarakenduse suhtluse parandused, toimima ühes konteineris
18.04	• Tagarakenduse ühendamine ThingsBoardiga ning ThingsBoard demoga
25.04	• Tagarakenduses stack overflow vea tõttu refaktoorimine, andmebaasi queryde vähendamine liidese muutmise abil • Tagarakenduse liidese muutmine, et oleks eraldi aadressid seadmete registreerimiseks ja andmete edastamiseks • Lõpuprojekti dokumentatsioon
02.05	• ThingBoardi telemeetria edastamine, vajalikud muudatused rakendustes • Lõpuprojekti dokumentatsioon
09.05	• Telemeetria edastamisega seotud parandused, testid • Lõpuprojekti dokumentatsioon

16.05	• Lõpuprojekti dokumentatsioon

Tabel 2. Ürgen Sõukandi logid.

Nädal	**Tegevused**
17.01	• Oauth2 tutvumine ja dokumentatsiooni läbitöötamine • Kubernetes Deployment skriptide koostamine ja dokumentatsiooni läbitöötamine
24.01	• Deployment skriptid UI-le, API-le, Postgres-ile-le, • Klastri modelleerimine ja lokaalse klastri ülesseadmise õppimine • Azure Kubernetes Cluster dokumentatsiooni lugemine
31.01	• Võimalikult väikse ressursikasutusega klastri ülesseadmise • Rakenduse terviklik toimimine lokaalses klastris • Pilve klastri välja lülitamise ja käivitamise automeerimine
07.02	• Hub koodi muutmine, et ta annaks infot enda alamseadmete kohta • UI autoriseerimisprobleemi lahendamine
14.02	• Sisselogimise ümbersuunamise probleemi lahendamine • Kogu lõputöö jaoks vajalike ticketite koostamine
21.02	• Ticketite korrastamine ja järjestamine, ajaplaneerimise pokker, • Jwt saamine sisselogimise kaudu • GraphAPI kasutamine varadele ligipääsemiseks
28.02	• Kubernetese klastri pilves käivitamine Github Actionsite abil • Laudade broneerimise süsteemi kalendri ürituste loomine MS GraphAPI abil
07.03	• Akri uurimine • Uue pilve klastri ülesseadmine väiksemate ressursinõuetega • Automatiseering selliselt et pilve klaster on töös ainult 08.00-18.00 tööpäevadel • Sertifikaatide genereerimine, et saaks Cluster API-le ligi • Kõikide projekti osade käivitamine pilves GitHub Actionsite abil

14.03	• Uue andmemudeli loomine, mis toetaks IoT-seadmete salvestamist andmebaasi • Andmebaasile kitsenduste lisamine, avastusrakenduse debuggimine • Erinevate Iot protokollide uurimine
21.03	• Avastusrakenduse probleemi Docker keskkonnas lahendamine • Hubilt saadud andmete salvestamine logina andmebaasi
28.03	• Event-based salvestamissüsteemi loomine • Salvestamisel tekkinud vea lahendamine • Teiste kontoris olevate IoT-seadmete ülesseadmise proov, ei võimaldanud meie tarkvaraga ühendamist
04.04	• Home Asssistant tarkvara uurimine • Event-based salvestamise refaktoorimine • Akri testimine lokaalses klastris • ThingsBoard Gateway kasutuse testimine
11.04	• Akri ülesseadmise diskussioon mentoritega • MQTT vahendaja ülesseadmine pilve klastrissebrokeri • Avastusrakenduse deployment failide loomine
18.04	• Probleemid client secretiga avastusrakenduses, nende lahendamine • Akri konfigureerimine jooksutamaks avastusrakendust • Proov Akrit pilve klastris jooksutada
25.04	• Akri jooksutamise probleemide lahendamine • Avastusskripti muutmine vältimaks avastuse lõputut ringi • Keskkonna muutuja sisestamine Akri abil avastusrakendusse
02.05	• ThingsBoard reeglistike katsetamine ja loomine • Akri käivitamine pilves Github Actionsite abil, ei saanud tööle
09.05	• Thingsboardi töölaudade koostamine ja katsetamine • Projekti dokumenteerimine
16.05	• Akri käivitamine GitHub Actionsite abil toimib, lisatud saladused klastri kohta • Projekti automaatse avastuse dokumenteerimine

Kokkuvõte

IoT-seadmete suuremahuline ülesseadmisega kaasneb mitmeid probleeme nagu suur ajakulu, erinevate protokollidega seadmed ning nende hilisem haldus ja koostöö. Töö eesmärgiks oli luua rakendus, mis aitaks neid probleeme lahendada.

Selleks kombineeriti vabavaralised rakendused Akri ning ThingsBoard autorite Spring Boot raamistikul kirjutatud avastus- ning tagarakendusega. Kogu lahendus konteineriseeriti ning juurutati Azure Kubernetes Service pilveteenuses. Kasutati agiilseid arenduse metoodikaid.

Nende tehnoloogiate kombineerimise tulemusena loodi süsteem, mis suudab IoT-seadmeid automaatselt avastada, kui need ühenduvad sõnumivahendajaga. Pärast avastamist käivitatakse avastusrakendus, mis registreerib seadme tagarakenduses ning hakkab edastama seadme telemeetriat haldusportaali. Viimases saab seadmeid siduda varadega ning koostada telemeetria põhjal reegleid kasutaja poolt soovitud toimingute käivitamiseks.

Rakendust kasutati Eesti Energias laudade broneerimise süsteemi prototüübi seadmete avastamiseks ja registreerimiseks. Kuna rakendus sai selle ülesandega hakkama, võib eesmärgi lugeda täidetuks. Selleks, et süsteem suudaks registreerida igat tüüpi seadmeid, tuleb teha mõningaid edasiarendusi, mis on töö analüüsis välja toodud.

Kasutatud kirjandus

[1] C. Petrov, „49 Stunning Internet of Things Statistics 2022 [The Rise Of IoT],“ Tech Jury, 26 Aprill 2022. [Võrgumaterjal]. Available: https://techjury.net/blog/internet-of-things-statistics/#gref. [Kasutatud 17 Mai 2022].

[2] P. Wegner, „IoT Analytics,“ 30 märts 2022. [Võrgumaterjal]. Available: https://iot-analytics.com/iot-market-size/. [Kasutatud 15 mai 2022].

[3] K. L. In Lee, „The Internet of Things (IoT): Applications, investments, and challenges for enterprises,“ *Business Horizons*, kd. 58, nr 4, pp. 431-440, 2015.

[4] J. S. Ken Schwaber, „The Scrum Guide,“ 11 2020. [Võrgumaterjal]. Available: https://scrumguides.org/scrum-guide.html. [Kasutatud 16 05 2022].

[5] Home Assistant, „Home page,“ [Võrgumaterjal]. Available: https://www.home-assistant.io/. [Kasutatud 17 Mai 2022].

[6] A. Z. a. A. A. S. H. Rahimi, „A Novel IoT Architecture based on 5G-IoT and Next Generation Technologies,“ *IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pp. 81-88, 2018.

[7] A. S. Imran Amin, „Wireless Technologies in Energy Management,“ %1 *Comprehensive Energy Systems*, Elsevier, 2018, pp. 389-422.

[8] MQTT, „MQTT: The Standard for IoT Messaging,“ 2022. [Võrgumaterjal]. Available: https://mqtt.org/. [Kasutatud 17 Mai 2022].

[9] Connectivity Standards Alliance, „Innovation & Adoption – Zigbee momentum in 2021,“ Connectivity Standards Alliance, 27 1 2022. [Võrgumaterjal]. Available: https://csa-iot.org/newsroom/innovation-adoption-zigbee-momentum-in-2021/. [Kasutatud 14 05 2022].

[10] G. C. Hillar, „Preface,“ %1 *MQTT Essentials - A Lightweight IoT Protocol*, Packt Publishing, 2017.

[11] HiveMQ, „MQTT Overview,“ 2022. [Võrgumaterjal]. Available: https://www.hivemq.com/mqtt/mqtt-protocol/. [Kasutatud 17 Mai 2022].

[12] Akri, „What is Akri?,“ Jaanuar 2022. [Võrgumaterjal]. Available: https://docs.akri.sh/. [Kasutatud 17 Mai 2022].

[13] R. C. Martin, Clean Architecture: A Craftsman's Guide to Software Structure and Design, Pearson, 2017.

[14] T. Hombergs, Get Your Hands Dirty on Clean Architecture, Packt Publishing, 2019.

[15] L. Gupta, „What is REST,“ 7 Aprill 2022. [Võrgumaterjal]. Available: https://restfulapi.net. [Kasutatud 16 Mai 2022].

[16] The ThingsBoard Authors, „ThingsBoard demo,“ 2022. [Võrgumaterjal]. Available: https://demo.thingsboard.io/login. [Kasutatud 16 5 2022].

[17] R. H. R. J. J. V. Erich Gamma, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, 1994.

[18] The ThingsBoard Authors, „What is ThingsBoard,“ 2022. [Võrgumaterjal]. Available: https://thingsboard.io/docs/getting-started-guides/what-is-thingsboard/. [Kasutatud 14 05 2022].

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Meie, Allan Päll ja Ürgen Sõukand

1. Anname Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Asjade interneti seadmete haldamise lahendus koos automaatse avastusega Eesti Energia AS näitel“, mille juhendaja on Tarvo Treier
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Oleme teadlikud, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autoritele.
3. Kinnitame, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

18.05.2022

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti

Lisa 2 – Allan Pälli eneseanalüüs

Lõpuprojekti tegime kahekesi olles kuueliikmelise tiimi osa. Kõikides lõputöö projekti osades panustasime mõlemad, kuid pean oma peamiseks panuseks:

- projekti arhitektuuri visandamine,
- projekti backlogi ülesannete kirjeldamine ning nimekirja haldamine,
- Github Actionsitega pideva integratsiooni protsessi tegemine,
- tagarakenduse ning avastusrakenduse koodi,
- IoT seadmete andmete simuleerimine lahenduse testimiseks.

Projekt oli huvitav mitmes aspektis – meile usaldati ettevõttes ühe lahenduse täielik katsetamine sealhulgas projekti skoobi ning suundade üle otsustamine, projekt sisaldas palju minu jaoks uudseid tehnoloogiaid ning seda sai ellu viia agiilse metoodika alusel.

Pärast meeskonnatöö projekti eelmisel semestril otsustasime kuue peale kirjutada kolm tööd – pean seda väga heaks valikuks kuna selliselt sai kahekesi minna ühes valdkonnas rohkem süvitsi ning lõputöö kirjutamine oli ka selgem. Samas oli kevadel endiselt väga hea, et siiski säilis meeskonnavaim kuuekesi ning arusaam, et vaatamata erinevatele lõputöödele, on meie projekt Eesti Energias endiselt ühine. Samuti andsime pidevalt üksteisele tagasisidet ning aitasime üksteise vigu otsida ning siluda olenemata sellest, kelle lõpuprojektiga tegu oli.

Lõpuprojekti osas olen rahul enda panusega ning väga hea koostööga, mis meil kahekesi tekkis. Eriti hea oli kogeda ka seda, kuidas hakkama saada olukorras, kus nõudeid ja tahtmisi on palju, aga tuleb valida, millele keskenduda. Tuli teha ka valikuid, mis muutsid meie plaane kategooriliselt ning see oli hea, et nii meil arendajatena, kui ka ettevõtte poolselt, oli nendele otsustele toetus.

Kõige keerulisemaks projekti juures peaksin CI/CD ning pilveteenuste koordineerimise protsesse. Kogu projekti raames oli meil kasutusel tehnoloogiad, mida otseselt õppetöös väga palju kas ei õppinud või ei kasutanud. Sinna hulka kuulub seadmetega kommunikatsioon, sõnumivahetuse protokollid, Kubernetes ja CI/CD protsessi

ülesseadmine, pilveteenuste kasutamine ning autentimise ja sisselogimise tegemine rakenduste vahelises suhtluses. Siia lisaks ka oma klastri ülesseadmine ning haldamine, millega tegeles rohkem küll Ürgen, aga tihti oli asju tarvis koos vaadata. Teiseltpoolt kasutasime ka uusi keeli ja raamistikke, mis oma põhimõtetelt küll sarnased - nt Java ja C#, kuid siiski nüanssides piisavalt erinevad, et nende kasutamisel kulus rohkem aega töö tegemisel. Projekti tegemine nõudis palju iseseisvalt ning koos õppimist. Aga see kõik oli seda väärt, kuna nende oskuste omandamine oli huvitav ning on kasulik ja muidugi ka tööandjate hulgas väärtustatud.

Väga kasulik oli ka jätkamine Scrum raamistikuga ning see, et kevadel usaldati meile toote juhtimine ning pidime ise ülesanded kirjutama ja neid Scrum metoodikaga haldama. Projekt õpetas palju ning motivatsiooni jätkus sellega lõpuni. Selle aja jooksul olen näinud, kuidas tudengiprojektid Eesti Energias on leidnud kas otse või kaudu reaalset rakendust. Seetõttu oli hea kuulda ettevõtte sees lahendust demonstreerides, et suvel plaanitakse teha meie töö alusel ka reaalne test suurema hulga seadmetega, et katsetada kontorilaudade broneerimise süsteemi.

Lisa 3 – Ürgen Sõukand eneseanalüüs

Enda põhilisteks panusteks lõputöö projektis tooksin välja:

- Pilvekeskkonna ülesseadmine
- Kuberneteses rakenduste orkestreerimine
- Andmemudeli väljatöötamine IoT-seadmete salvestuseks
- Event-põhise salvestussüsteemi rakendamine
- Lahenduse juurutamine pilvekeskkonnas
- Akri konfigureerimine ja ülesseadmine

Tehnoloogilises mõttes oli minu jaoks kõige huvitavam, kuid ka keerukam rakenduste orkestreerimine Kubernetes keskkonnas ning selle pilveteenuses juurutamine. See nõudis palju juurde õppimist, kuna ma polnud sellega ülikooli ajal varem kokku puutunud. Mulle meeldis väga tegeleda ka IoT-seadmete katsetamisega, vahest ka liiga palju, nii et teised kohustused jäid tahaplaanile.

Kõige uhkem olen selle üle, et suutsime ise rakendada Akri ning avastusskripti enda klastris, sest esialgu pidi see funktsionaalsus tulema Eesti Energia poolt, kuna tegu on üsna kõrgetasemelise kontseptsiooniga rakendusega.

Mulle meeldis, et ettevõte andis meile lõputöö projekti raames vabad käed. See andis tugeva omanditunde, kuna kogu lahendus oli meie disaini ning otsuste järgi loodud. Kuigi pidime majandama iseseisvalt, sujus korraldus väga hästi, sest Allan on suurepärane planeerija.

Meie loodud lahendus hõlmas endas palju erinevaid tehnoloogiaid ning eeldas palju integratsioone, mistõttu oli see väga heaks õppimise allikaks. Lisaks oli see võimaluseks leida enda jaoks huvitav arendussuund. Leidsin, et mulle meeldib tegeleda IoT süsteemide ning pilvetehnoloogiatega ja asun ka selles valdkonnas Eesti Energias tööle.