TALLINNA TEHNIKAÜLIKOOL

INSENERITEADUSKOND

Tartu kolledž

RAKENDUS JAHUTUSSÕLME TÖÖ OPTIMEERIMISEKS

APPLICATION FOR OPTIMISING THE OPERATION OF A COOLING PLANT

BAKALAUREUSETÖÖ

Üliõpilane: Lauri Tigasson

Üliõpilaskood: 154570NDFR

Juhendajad: Sven Oras, Lektor

Silver Saar, Automaatikainsener

Tartu 2020

AUTORIDEKLARATSIOON

Olen koostanud lõputöö iseseisvalt.
Lõputöö alusel ei ole varem kutse- või teaduskraadi või inseneridiplomit taotletud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

"....." ..... 2020.

Autor: .....
/ allkiri /

Töö vastab bakalaureusetöö/magistritööle esitatud nõuetele

"....." ..... 2020.

Juhendaja: .....
/ allkiri /

Kaitsmisele lubatud

"....."..... 2020 .

Kaitsmiskomisjoni esimees .....
/ nimi ja allkiri /

TalTech Tartu kolledž

LÕPUTÖÖ ÜLESANNE

Üliõpilane: Lauri Tigasson, 154570NDFR

Õppekava, peaeriala: NDFR14/15 - Küberfüüsikaline süsteemitehnika

Juhendaja(d): Lektor, Sven Oras, 620 4807

Automaatikainsener, Silver Saar, 5620 3286

Lõputöö teema:

Rakendus jahutussõlme töö optimeerimiseks

Application for optimising the operation of a cooling plant

Lõputöö põhieesmärgid:

1. Luua rakendus jahutussõlme töö optimeerimiseks
2. Kirjeldada rakenduse ja selle osade tööpõhimõtteid

Lõputöö etapid ja ajakava:

Nr	Ülesande kirjeldus	Tähtaeg
1.	Rakenduse tööks vajalike seadmete ja tarkvarade ettevalmistamine	08.03
2.	Rakenduse programmi valmistamine ja riistvara paigaldamine hoonesse	19.04
3.	Rakenduse katsetamine ja tulemuste võrdlemine varasemate andmetega	10.05

Töö keel: eesti keel **Lõputöö esitamise tähtaeg:** " 25" mai 2020.a

Üliõpilane: Lauri Tigasson ..... "....."..... 2020.a
/allkiri/

Juhendaja: Sven Oras ..... "....."..... 2020.a
/allkiri/

Juhendaja: Silver Saar ..... "....."..... 2020.a
/allkiri/

Programmijuht: Helle Hallik ..... "....."..... 2020.a
/allkiri/

SISUKORD

SISSEJUHATUS

Olemasolevat tehnoloogiat ja Internetti ühenduvaid seadmeid sidudes, liites nende otstarbed ning seadmed andmesidelt, võib luua süsteemi, mis muudab selle osade töö efektiivsemaks. Selles töös kavatseb autor luua rakenduse, mis erinevate pilveteenuste ning seadmete töö ja suhtluse tulemusel üritab muuta jahutussõlme töö efektiivsemaks. Lõputöö peamiseks eesmärgiks on luua REST API-sid kasutav rakendus, mis ajastaks büroohoonet teenindava jahutussõlme tööd nii, et jahutussõlme komponendid teeks tööd vaid vajadusel ning kui võimalik, siis madalama rahalise kuluga. Kuna jahutussõlmes jahutatavat vett kasutatakse büroohoone kontoriruumide jahutamiseks, siis peab autor vajalikuks, et loodav rakendus peaks komponentide töö ajastamisel tähtsaks ruumi kasutaja mugavust ehk rahalist kulu alandades ei kaoks jahutussõlme mõte ja funktsionaalsus. Lõputöö lisaeesmärgiks on kirjeldada API-de ning täpsemalt REST API-de tööpõhimõtteid, et lugeja saaks parema aimduse kuidas neid üldiselt kasutada ning milleks autor neid töös käsitletavas rakenduses teeb.

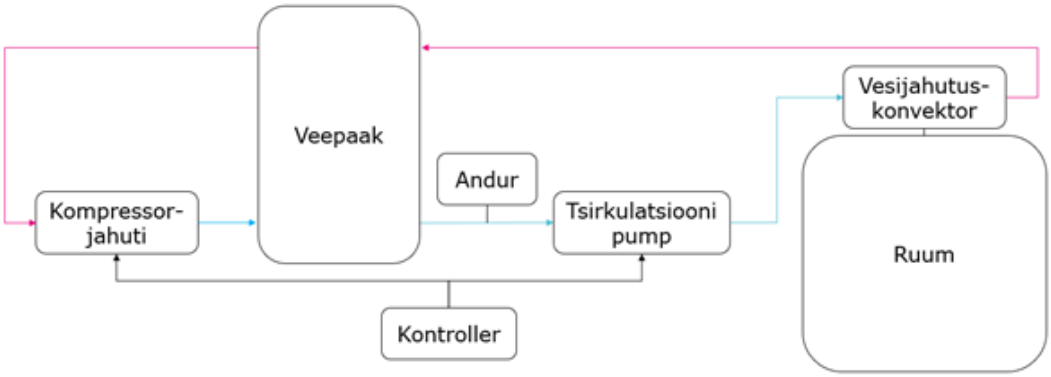
Autor valis oma lõputöö teemaks just REST API-sid kasutava veebirakenduse loomise, sest tundis huvi REST tüüpi API-de kasutamise viiside vastu ning soovis luua nende funktsionaalsust kasutava rakenduse. See mõte, et rakendus jahutussõlme töö juhtimist hakkaks mõjutama, tuli alles peale teemavalikut. Teemal on autori arvates tähtsust, sest selline rakenduse loomine näitab, et maailmas on olemas erinevad seadmed ja tarkvarad, mida ühendades on võimalik luua midagi uut või olemasolevat täiustada kui neid vaid kasutada.

Lõputöö keskendub Tartus Sõbra 54 aadressil asuva büroohoone maa-aluse parkla korrusel oleva jahutussõlme töö mõjutamisele, et sama hoone ruumide jahutusele kuluvat energiatarvet vähendada. Enne kui autor oli oma lõputööks vajalike tegevustega jahutussõlme ümber seadistanud, töötas jahutussõlm pidevalt, ilma et selle tööd ajastataks.

Et lugejal oleks lõpplahendusest kergem aru saada, siis lõputöö esimeses pooles selgitab autor taustainfoks API-de ning nende tööks vajalike meetodite ja elementide toimimist. Järgmiseks selgitab autor missuguses keskkonnas rakendus luuakse ja kuidas programmeeritakse täitma ettenähtud funktsiooni. Lõputöö viimases pooles annab autor lugejale rakenduse programmi sisust ülevaate ning kirjeldab millistel põhimõtetel ja tingimustel rakendus jahutussõlme komponente juhtima hakkab. Viimaseks tõestab autor, et valmistatud rakendus on muutnud jahutussõlme töö efektiivsemaks ja vähem kulukamaks.

1 VARASEMA KIRJELDUS

Töös käsitletavas büroohoones oli enne lahenduse sisse viimist jahutussõlme ning ruumi kliimatehnika töökäik selline, kus sõlmes jahutatakse vett kompressor-jahuti abil, mis kasutab vee jahutamiseks välisõhku. Tsirkulatsioonipump liigutab jahutatud vee edasi vähese soojusisolatsiooniga paaki, millest väljuva toru juures on andur vee temperatuuri mõõtmiseks. Vastavalt hooneautomaatikahalduri poolt ettekirjutatud seadepunktile jahutab kompressor-jahuti vajadusel vett ehk jahutab nii, et anduri mõõdetud temperatuur vastaks seadepunktile. Paagist pumpab tsirkulatsioonipump vee edasi hoone ruumides paiknevatesse vesijahutus-konvektoritesse (inglise keeles *fan coil unit*), kus vesi voolab läbi soojusvaheti, et seda ning seeläbi ümbritsevat õhku jahutada. Et ruumi õhku jahutada puhub soojusvaheti juures olev ventilaator jahutatud õhu ruumi. Vastavalt ruumis paiknevale Siemensi nutitermostaadi temperatuuri näidule ning ruumi kasutaja poolt ettekirjutatud seadepunktile ventilaator puhub või mitte. Kuna vesijahutus-konvektorisse pumbatud vesi on soojenenud peale soojusvaheti jahutamist, siis pumbatakse see jahutussõlmes olevasse veepaaki tagasi ning uuesti läbi kompressor-jahuti, et seda jahutada. Seega on paak alati vett täis ja tegu on kinnise süsteemiga, kus vett juurde ei tule ega ka süsteemist välju. Enne lahenduse sisseviimist oli tsirkulatsioonipump kogu aeg sisse lülitatud ehk tegi tööd pidevalt ning seiskus ainult kui kadus toide, tekkis viga või hooneautomaatikahaldur andis selleks käsu. Kompressor-jahuti töötas vaid juhul, kui vee temperatuuri oli vaja alandada seadepunktini. Varasema lahenduse kirjeldust illustreerib Joonis 1.1



Joonis 1.1 Lihtsustatud joonis jahutussõlme toimimisest

1.1 Rakenduse eesmärk

Töös kirjeldatav rakendus keskendub jahutussõlme komponentide töö ajastamisele. Rakendus peab programmis ettenähtud tingimustel mõjutama kompressor-jahuti tööd

ning millal tsirkulatsioonipump pumpab paagist vett edasi ruumides olevatesse vesijahutus-konvektoritesse. Lahenduse rakendamisel peab kompressor-jahuti jahutama vett ja tsirkulatsioonipump pumpama selle edasi ruumidesse ehk komponendid peavad kulutama energiat ainult juhul, kui elektribörsi hind on soodne ja rakendus aimab ette, et jahutust võib tulevikus vaja minna või kui hetkelise jahutusnõudluse tõttu on tarvis kindlasti ruume jahutada. Kui ruumid vajavad jahutust, siis komponendid peavad tööd tegema olenemata hinnast. Võttes arvesse elektribörsi hinda ning jahutusvajadust peab rakendus suutma ajastada jahutussõlme tööd ning alandada selle läbi energia ja rahalist kulu, kuid seda tehes tuleb ruumi kasutaja mugavust säilitada.

2 SEADMETE KIRJELDUS

Töös käsitletav rakendus vajab, et jahutussõlme komponentide andmepunktid, mida töö programmilises mõjutatakse, oleksid kättesaadavad läbi Interneti mingisugusest andmebaasist, millest rakendus saab neid küsida. Andmepunkt ise on tavamõistes jupp informatsiooni, mis võib selles töös olla näiteks andurilt saadud temperatuur või kompressor-jahuti tööolek (Rouse, data point, 2012). Ühed töös kasutatavad andmepunktid on kompressor-jahuti ning tsirkulatsioonipumba töökäsud, mida mõjutatakse, et neid sobival ajal käivitada ja peatada. Need andmepunktid loodi programmeeritavas loogikakontrolleris vastavalt programmile. Andmepunktide hoiustamiseks valis autor Siemensi Climatix IC pilveteenuse. Et seda pilveteenust oleks võimalik kasutada valiti ka jahutussõlme juhtimiseks Siemensi Climatix tüüpi kontroller, sest see on selle pilvega ühilduv. Andmepunktide pilvega sidumiseks oli tarvis kontrolleri sätetes sisestada pilvekasutajapõhine võti, pilves kinnitada kontrolleri sidumine ja teha pilvepoolne kontrolleri seadistus. Kuna pilv on ligipääsetav interneti kaudu, siis oli ka tähtis, et kontrolleril oleks olemas ühendus internetiga. Lisaks kontrollerile kasutatakse töös Siemensi RDG100KN nutitermostaate, mille kogutuid andmeid rakendus kasutab jahutussõlme juhtimisel. Ka need nutitermostaadid ühendas autor Climatix IC pilvega ning tegi sarnased seadistused seadme pilvega sidumiseks nagu kontrolleri puhul ning tagas, et termostaatidel oleks ühendus Internetiga.

2.1 Pilve kirjeldus

Climatix IC on pilveteenus mõeldud teatud tüüpi Siemensi seadmete andmete kogumiseks, analüüsimiseks ja jälgimiseks. Pilvele ligipääsuks on lisaks veebibrauseri kasutamise võimalusele Siemens loonud ka REST API, läbi mille on võimalik pilve kogutud andmeid kasutada ka pilvevälistes rakendustes. (Siemens, kuupäev puudub) Kuna Siemensi loodud API võimaldab luua andmeside pilve ja töös käsitletava rakenduse vahel, siis võimaldab see API ka seeläbi töös käsitletaval rakendusel mõjutada jahutussõlme komponente. Seega on Climatix IC kasutamisega rahuldatud peatüki alguses kirjeldatud rakenduse vajadus, mis nägi ette, et jahutussõlme komponentide andmepunktid peaksid olema kättesaadavad läbi interneti mingisugusest andmebaasist. Andmepunktide hoiustamiseks valis autor Climatix IC teiste alternatiivsete andmebaaside asemel, sest juba oli olemas Climatix IC kasutaja, kogemus selle pilveteenuse kasutamisega ning ka Climatix tüüpi kontroller, millega luua ühendus pilvega.

3 API

Et mõista kuidas töös kirjeldatav rakendus on võimeline küsima infot erinevatest pilvedest, on tarvis esmalt mõista mis on API ja kuidas see töötab. Seetõttu otsustas autor oma töös anda lühida ülevaate erinevatest olemasolevatest API-dest ning selgitada nende töökäiku.

3.1 API selgitus

API ehk *Application Programming Interface* on tarkvaraliides, mille ülesanne on vahendada infopäringuid kahe tarkvara vahel. API-de kasutusvaldkond on lai – näiteks kasutatakse neid operatsioonisüsteemides, andmebaasi süsteemides ning veebipõhistes süsteemides. (Davis, 2019) API-t nimetatakse veebipõhiseks API-ks, kui suhtlus tarkvarade vahel toimub läbi interneti kasutades veebispetsiifilisi protokolle nagu näiteks HTTP (Pedro, 2017). Veebi API-d võivad olla kliendi- või serveripoolsed. Kliendipoolne tähendab, et API tööks vajaliku koodi jooksutamine kliendi ja serveri vaheliseks suhtluseks või kliendi sisemiste funktsioonide läbi viimiseks toimub kliendi poolel, näiteks veebibrauseris. Serveripoolne tähendab, et API koodi jooksutatakse serveris. (Cloudflare, kuupäev puudub) Kliendipoolseid API-sid kasutatakse laialdaselt veebibrauseri erinevate aspektide juhtimiseks. (Introduction to web APIs, 2019) Serveripoolsed API-d leiavad aga rohkem kasutust serveri ja kliendi vahelise suhtluse ehk päringute saatmiste vahendamisel. (Introduction to the server side, 2019) On oluline aru saada, et mõlemat sorti API-d luuakse serveri või kliendi omanike poolt. Seega on API ehitatud teenindama mingit kindlat serverit või klienti ning on osa sellest. Selles töös käsitleb autor edaspidi serveripoolseid API-sid ehk API-sid, mis saavad kliendilt info ja vastavalt saadud infole küsivad serverilt vastust, mis edastatakse tagasi kliendile.

Autor selgitab mõningaid mõisteid, et lugejal oleks teemast kergem aru saada. Kliendiks on tavaliselt mingisugune rakendus, näiteks veebibrauser, mis töötab kasutaja arvutis, mobiiltelefonis või mõnes teises (Network Client, kuupäev puudub). Serveriks nimetatakse arvutit või süsteemi, mis jagab läbi võrgu andmeid, teenuseid, programme või ressursse klientidele (Paessler AG, kuupäev puudub).

Lugejale selgituseks toob autor näite elust enesest – kuidas kliendi, serveripoolse API ja serveri suhet võib võrrelda restoranis toidu tellimisega. Klient läheb restorani ja tal on soov menüüst tellida toitu. Kliendi soovi vaatame kui kliendi päringut serveri ehk selles näites köögi poole. Et see päring serverisse jõuaks on vaja API-t ehk kelnerit, kes küsib kliendilt missugust toitu ta tahab. Selle info edastab API serverile ehk kelner köögile ja

vastavalt päringus täpsustatud infole antakse tagasi ka vastava sisuga vastus ehk see toit, mille klient tellis.

API-sid saab lisaks klassifitseerida kolme parameetri järgi: omandiõiguse tüüp, kommunikatsiooni tase ja veebiteenuse API-d. Veebiteenuse API-deks nimetatakse API-sid, mis kasutavad veebiaadresse, et Internetis tagada ligipääsu teenustele. Veebiteenuse API-de klassifikatsioon toimub API ehitusel kasutatud kommunikatsiooni- ning käitumisviisi järgi ning need jagunevad neljaks: SOAP, XML-RPC, JSON-RPC ja REST. (Nehra, 2019)

SOAP (*Simple Object Access Protocol*) API-d kasutavad SOAP protokolli, mis kasutab ainult patenteeritud XML andmeformaati oma andmevahetuses. Andmevahetus on protokollis turvaline ja töökindel. (Nehra, 2019)

XML-RPC (*Extensible Markup Language – Remote Procedure Calls*) API-d töötavad protokollil, kus kasutatakse täpsustatud XML andmeformaadi varianti oma andmevahetuses. Lisaks kasutab protokoll minimaalset ribalaiust andmevahetusel ja on ehituselt palju lihtsam kui SOAP protokoll. (Nehra, 2019)

JSON-RPC (*JavaScript Object Notation – Remote Procedure Calls*) API-d kasutavad protokolli, kus andmevahetus toimub JSON andmeformaadis (Nehra, 2019).

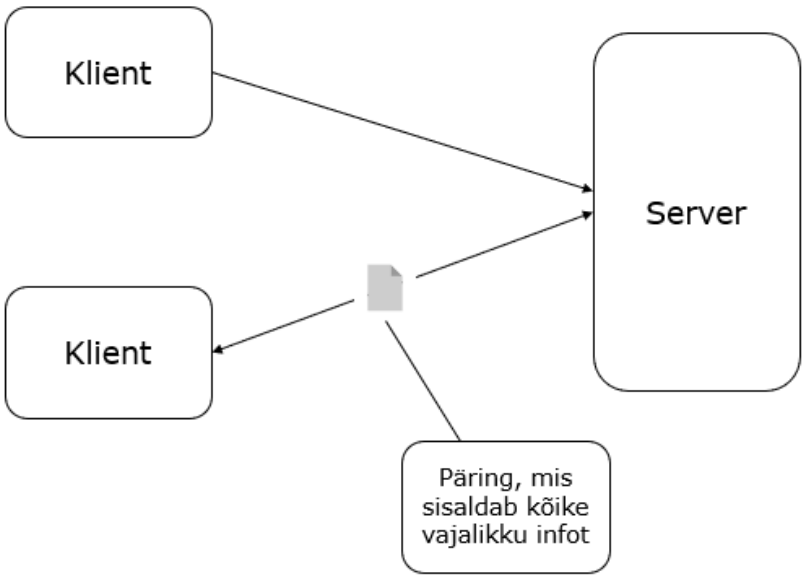
4 REST API

REST ehk *Representational State Transfer* on arhitektuuri stiil, mida kasutatakse veebiteenuste arendamisel. Et veebiteenus (sealhulgas APId) oleks REST tüüpi, peab see kinni pidama kuuest arhitektuurilisest kitsendusest. (Rouse, RESTful API (REST API), 2020) Nendest kuuest kitsendusest keskendub autor antud töö raames ainult kolmele, sest leiab, et nendest piisab REST API tööpõhimõtete selgituseks.

4.1 REST API ehitus

Esiteks peavad klient ja server olema üksteisest eraldatud nii, et ühes muudatusi tehes teises muudatusi ei toimu. Näiteks kui kliendiks on mobiilirakendus või veebibrauser ja selles teha muudatusi, siis serveri andmebaasi struktuuris ega disainis muudatusi ei toimu. Sama kehtib ka vastupidises olukorras, kus serveris tehtud muudatused ei mõjuta klienti. (What is REST API design?, kuupäev puudub) See tähendab, et API on võimeline töötama eri sorti klientidega, mis võivad iseseisvalt teha muudatusi.

Teiseks peab API olema olekuta. See tähendab, et iga API päring kliendilt serverile peab sisaldama piisavalt infot, et päringut oleks võimalik läbi viia ilma et vajaks mujalt lisainfot, näiteks serverilt sisendit. (What is REST API design?, kuupäev puudub) See sarnaneb posti saatmisega, kus ümbrikul peab olema märgitud kirja saaja ja tema aadress. Et post kohale jõuaks ei pea kirja saaja enam andma selleks lisainfot, sest kõik vajalik on juba ümbrikul olemas. Seda kitsendust illustreerib Joonis 4.1.



Joonis 4.1 Klientide suhtlus serveriga

Kuna kogu vajalik info on päringus olemas on API töökindlam, sest ei pea näiteks kliendi tuvastuseks tegema serverile lisapäringuid. (What is REST API design?, kuupäev puudub)

Kolmandaks peavad REST arhitektuuris ressursid olema üksteisest eristatavad URI-de ehk *Uniform Resource Indentifier*-ite kaudu (REST Architectural Constraints, kuupäev puudub). URI-d on identifikaatorid, mis viitavad ressurssidele andes neile nime, et neid oleks võimalik eristada (Miessler, 2019). Näiteks võib ühe ressursi URI olla „näide.ee/ressurss1" ja teise ressursi URI „näide.ee/ressurss2".

REST arhitektuuris võib ressursiks olla igasugune tükk infot nagu näiteks pilt, dokument, kogum teistest ressurssidest või selles töös ka kompressor-jahuti töörežiimi andmepunkt (Fielding, 2000).

Käesolevas töös kirjeldatakse just REST API-de tööpõhimõtteid, sest töös käsitletavas rakenduses kasutatakse samuti REST tüüpi API-sid mitmest erinevast pilvest andmete pärimiseks ja ka andmete uuendamiseks ning seetõttu soovib autor, et lugejal oleks parem arusaam kuidas REST API päringuid tehakse ning mis on nende sisu.

4.2 HTTP meetodid

Et RESTful API-l oleks võimalik teha päringuid serverile mingisuguse ressursi kohta peab olema päringus täpsustatud mida ressursiga teha soovitakse. Selleks kasutatakse tüüpiliselt, ja ka selles töös, hüperteksti edastusprotokolli ehk HTTP päringumeetodeid. (What is REST?, kuupäev puudub) Suur osa sellest miks REST arhitektuuris peavad ressursid olema eristatavad URI-de kaudu on see, et API saaks kasutada HTTP-d ja selles tehtavaid CRUD (akronüüm inglise keelsetest sõnadest *create*, *read*, *update*, *delete*) toiminguid. (Using HTTP Methods for RESTful Services, kuupäev puudub)

Create viitab HTTP meetodile POST. Kui API teeb serverile päringu kasutades päringumeetodit POST, siis see tähendab, et serverisse luuakse päringu kehas määratud asukohale ja sisuga uus ressurss (HTTP Methods, kuupäev puudub)

Read viitab meetodile GET, mis tähendab, et kui API teeb serverile päringu kasutades päringumeetodit GET, siis vastuseks antakse esitus sellest ressursist. See tähendab, et ressurssi serverist ära ei võeta, aga kliendile antakse kogu info, mis selle ressursi kohta serveris on. (HTTP Methods, kuupäev puudub)

Update viitab päringumeetodile PUT, mis asendab serveris paikneva ressursi sisu sellega, mis päringu kehas määrati. Juhul kui serveris pole olemas päringus kirjeldatud ressurssi võib API luua uue ressursi sarnaselt POST meetodile. (HTTP Methods, kuupäev puudub)

Delete viitab meetodile DELETE, mida kasutatakse ressursside kustutamiseks serverist. (HTTP Methods, kuupäev puudub)

On olemas ka meetodid OPTIONS, HEAD, CONNECT, PATCH ja TRACE, kuid neid selle töö raames ei kirjeldata, sest need meetodid ei leia töös kasutust.

4.3 Päringu ehitus

Kuna REST arhitektuuris peab päring sisaldama piisavalt infot, et see oleks iseseisvalt ehk ainult kliendi poolt antud info põhjal läbi viidav, siis selleks peab päringul olema lõpp-punkt, meetod, päised ja keha, millest iga osa kannab vajalikku infot. Päringu lõpp-punktiks kutsutakse URL-i kuhu päring saadetakse. Lõpp-punkti võib jaotada aga veel kolmeks osaks: juurlõpp-punktiks, rajaks ja päringu parameetriteks. Juurlõpp-punkt on selle API alguspunkt, mille poole pöördutakse. Näiteks „api.näide.ee" võiks olla API alguspunkt. Rada määrab lõpp-punktis ära selle ressursi asukoha ehk URI, mille kohta tahetakse pärida. Näiteks võib selleks olla „/ressursid/ressurss1". Seega, kui on soov küsida infot „ressurss1" kohta, siis tuleb päring saata aadressile „https://api.näide.ee/ressursid/ressurss1". Et teada saada millised rajad on kasutajale kättesaadavad, tuleks uurida API dokumentatsiooni enne kasutamist. Päringu parameetrid on lõpp-punkti viimane osa, kuid pole tehniliselt osa REST arhitektuurist. Sellegipoolest kasutavad paljud API-d päringu parameetreid oma päringutes, sest parameetrid annavad võimaluse kasutada päringus võtme-väärtus paare. Päringus algavad parameetrid küsimärgiga (?) ja peale seda on iga parameetripaar eraldatud ampersandiga (&). Näiteks nii: „?päring1=väärtus1&päring2=väärtus2". (Liew, 2018)

Lisaks lõpp-punktile peab päringus olema kirjeldatud meetod. Meetodist oleneb, mida ressursiga tehakse. Erinevaid HTTP meetodeid kirjeldas autor peatükis 4.2.

Kolmandaks on REST API päringus määratud päised ehk HTTP omadus-väärtus paarid. Päiste ülesanne on kanda infot. Selleks võib olla näiteks info, et päringu keha sisu on JSON andmeformaadis või võib see sisaldada serveris olevatele andmetele ligipääsuks vajalikku tunnusvõtit. (Liew, 2018)

Kui soovida, et iga suvaline REST API kasutaja ei saaks serveris olevat infot koguda või muuta, saab API-des kasutada autentimist ehk isiku tõendamist, autoriseerimist ehk volitamist ja tunnusvõtit ehk *token*'it. Autentimisel kontrollitakse kasutajanime ja salasõna korrektsust või et kas tunnusvõti on allkirjastatud ja kehtiv. Autoriseerimisel kontrollitakse aga kas kasutaja on volitatud ligipääsema või muutma seda ressurssi, mis on päringus määratud. Kasutaja võib olla näiteks volitatud küsima infot ressursi kohta GET meetodiga, aga mitte muutma sisu PUT meetodiga. Tunnusvõtmeid, mida API

turvalisusel saab kasutada, on mitmeid tüüpe, kuid selle töö lahenduses kasutatakse JSON *Web Token*'it (JWT). JWT on JSON objekt, mis on kodeeritud binaarkoodist tekstiks ja on allkirjastatud kasutades kas sümmeetrilist või ebasümmeetrilist krüpteerimist (Gilling, 2017). Esimesel juhul toimub krüpteerimine ja dekrüpteerimine sama võtmega ning teisel juhul kasutatakse nendeks tegevusteks erinevaid võtmeid – avalik võti krüpteerimiseks ja salajane võti dekrüpteerimiseks (Barker, 2016). JWT võib sisaldada tunnusvõtme tegija kohta infot, ajahetke kuna tunnusvõti väljastati ja kuna see aegub. Tunnusvõti paigutatakse päringus autoriseerimise päisesse. (Gilling, 2017)

Viimasena peab päringus olema kirjeldatud keha ehk info, mida tahetakse serverisse saata, kuid keha peab olema kliendi päringus kirjeldatud ainult juhul, kui kasutatakse meetodit, mis vajab lisaks infot sellele, mida juba eelnevalt päringus lõpp-punktis ega päises pole kirjeldatud. Nendeks meetoditeks on POST, PUT, PATCH ja DELETE. Näiteks kui tahta kasutada meetodit POST ehk luua uus ressurss, peab olema kehas täpsustatud uue ressursi sisu. Ressursi asukoht on aga juba eelnevalt määratud lõpp-punktis. Teiste meetodite kasutamisel peaks päringu keha tühjaks jätma. (Liew, 2018)

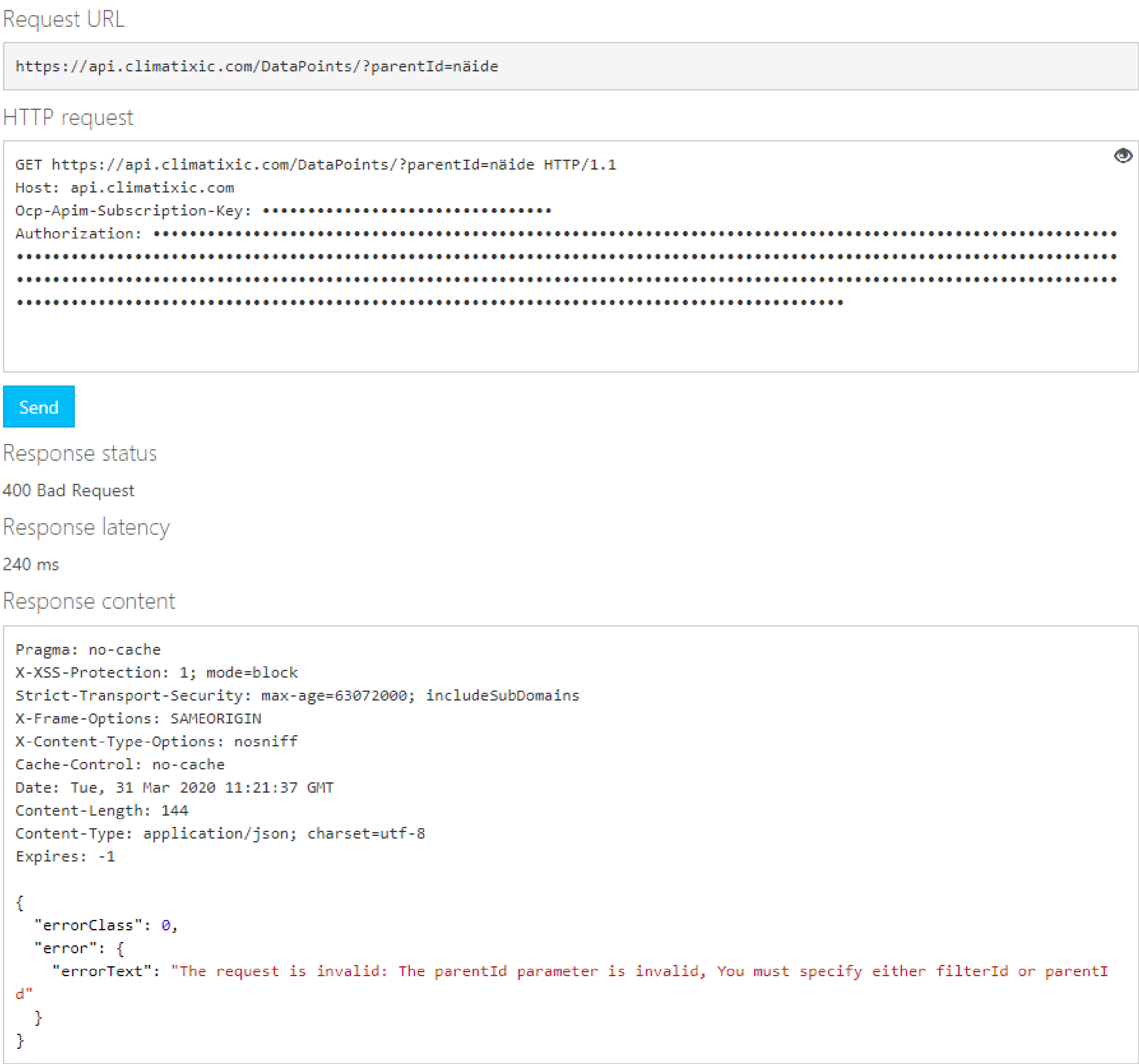
4.4 Päringu vastus

Kui klient on päringu saatnud serverile, siis peab serverilt ka vastus tulema. Olenevalt kasutatud HTTP päringumeetodist tuleb serverilt ka erinev HTTP vastus. Näiteks kui kasutades GET meetodit leitakse serverist päringus kirjeldatud ressurss, siis peab API tagastama serverilt HTTP koodi 200 (*OK*) koos vastuse kehaga. Kui aga ressurssi serverist ei leitud, siis tagastatakse HTTP kood 404 (*Not Found*) ning valesti vormistatud päringu puhul tagastatakse HTTP kood 400 (*Bad Request*). Kliendi poolt saadetud POST meetodit sisaldavale päringule tagastab õnnestunud pärimisel API serverist HTTP koodi 201 (*Created*) ja vastuse keha, mis sisaldab päringu staatust, viidet uuele ressursile ja asukoha päist. (HTTP Methods, kuupäev puudub)

Kui vastus tagastatakse HTTP koodivahemikus 100 – 199 tähendab see, et vastus on informatiivne. Näiteks kood 100 (*Continue*) tähendab, et senini on päringu saatmisel kõik korras ja et jätkataks päringu saatmist või et päringu vastust ignoreerida, kui see on juba ära saadetud. Vastuse koodivahemik 200 - 299 näitab, et päring oli edukas. Kui kood on vahemikus 300 – 399, siis viitab see sellele, et serveris on toimunud muudatused ja päringut suunatakse kuhugi. Vahemik 400 – 499 näitab, et klient on teinud vea päringu saatmisel, näiteks kui üritab serverist leida ressurssi, mida pole olemas. Ja viimaks, kui HTTP vastus on koodivahemikus 500 – 599, siis see tähendab, et server ei oska päringut või päringu meetodit käsitseda või et serveri siseselt esineb mingi probleem. (HTTP response status codes, 2020)

4.5 Näidispäring

Peatüki kokkuvõtteks toob autor näite Siemensi Climatix IC API-le päringu koostamise kujul. Päringu koostamine viiakse läbi Climatix IC API arendajaportaalis ehk veebilehel, mis on mõeldud Climatix IC pilve poole päringute tegemise lihtsustamiseks. Vaadates Joonis 4.2 HTTP *request* osa on näha tervet päringu sisu, mis API vahendusel pilve poole saadetakse. Päringus on määratud lõpp-punkt „https://api.climatixic.com/DataPoints/?parentId=näide", mis näitab seda, et soovitakse teha midagi ressursside kollektsioonis nimega *DataPoints* ja päringu parameeter *parentId* täpsustab, et päringu vastuses peaks olema ressurss või ressursid mille *parentId* = näide. Järgmiseks on päringus määratud meetod GET, seega tagastab API kogu info, mis kollektsioonis *DataPoints* selle ressursi kohta leida on. Rida „Host: api.climatixic.com" viitab päringu juurlõpp-punktile ning näitab, et pöördutakse Climatix IC API poole. *Ocp-Apim-Subscription-Key* ja *Authorization* on päised, mis on osa turbest. Esimene neist sisaldab API võtit, mis on seotud API kasutaja õigustega, on igal kasutajal erinev ja määrab kui palju päringuid on lubatud kasutajal pilve poole teha. *Authorizaton* sisaldab aga JWT-d, mis luuakse mitte arendajaportaali vaid Climatix IC pilve kasutajanime ja parooli sisestamisel. See tunnusvõti määrab päringus ära millele on volitust päringut saata ehk kui see ressurss, mille kohta kasutaja päringu saadab, on talle pilves omistatud ja tal on seal õigused sellega ümberkäimiseks, siis tunnusvõti autoriseerib kasutaja ka selle kohta API-ga pärima. Tunnusvõti ei jää kehtima igavesti ning selle peab tulevikus uuesti genereerima. Selles näites tehtud päringu vastuseks tuli serverist HTTP vastusekood 400 (*Bad Request*) ehk päring oli valesti vormistatud. *Response content* lahtris on näha, et vastus on JSON formaadis ning et parameeter nimega *parentId* ei ole sobival kujul. (Siemens AG, kuupäev puudub)



Joonis 4.2 Climatix IC API arendajaportaalis näidispäringu tegemine

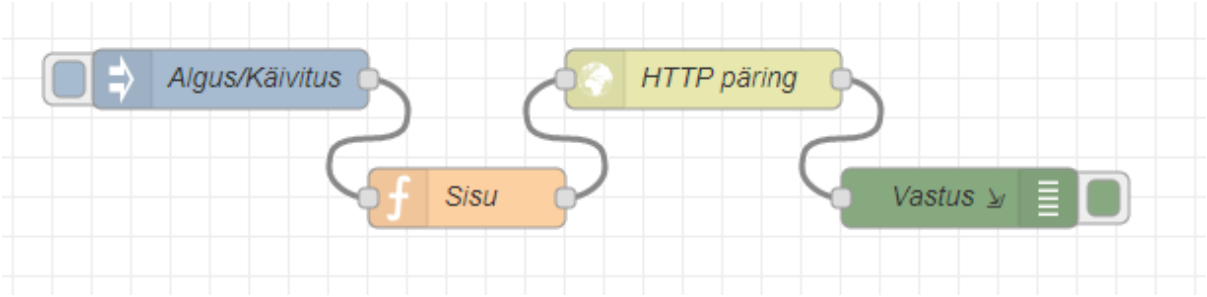
Node-RED-is valmistatud rakenduse programmilisel osas kasutatakse REST API-sid ning eelmainitud päringute saatmise põhimõtteid, et küsida infot erinevatelt pilvedelt ning saata käsklusi jahutussõlme komponentidele läbi Climatix IC pilve.

5 KESKKOND RAKENDUSE PROGRAMMEERIMISEKS

Selles peatükis kirjeldab autor programmeerimiskeskkonda, mille ta valis rakenduse programmeerimiseks, ja keskkonda kus programmi jooksutatakse ning põhjendab oma valikuid. Lisaks annab autor lühida ülevaate mõningatest alternatiividest mõlema valiku puhul.

5.1 Node-RED

Node-RED on visuaalseks voolupõhiseks programmeerimiseks (*flow-based programming*) mõeldud tööriist, mis pakub brauseripõhist redigeerimiskeskkonda riistvaraseadmete, API-de ja võrguteenuste sidumiseks. Node-RED on ehitatud Node.js põhjale. (Node-RED-i kodulehekülg, kuupäev puudub) Node.js on avatud lähtekoodiga *runtime* ehk tööaja keskkond võrgu- ja serveripoolsete rakenduste JavaScriptis arendamiseks ja jooksutamiseks (Node.js - Introduction, kuupäev puudub). See tähendab, et Node-RED-is programmeerimine toimub JavaScript programmeerimiskeeles. Voolupõhine programmeerimine tähendab, et rakenduse käitumist kujutatakse Node-RED-is sõlmede võrgustikuna. Sõlm on selline ettenähtud funktsionaalsusega osa programmist millele antakse infot, selle infoga tehakse midagi sõlme sees ja saadetakse see edasi. See kuidas andmed liiguvad sõlmede vahel sõltub sõlmede võrgustiku ehk voolu ehitusest (Joonis 5.1). (About, kuupäev puudub) Node-RED-is saab kasutaja luua sõlmi, voolusid ning salvestada neid teeki, et neid taaskasutada või teistega jagada (Node-RED-i kodulehekülg, kuupäev puudub).



Joonis 5.1 Sõlmedest koosnev võrk Node-RED keskkonnas

Node-RED-i on võimalik jooksutada kolmes erinevas keskkonnas:

Arvutis

Arvutis Node-RED-i jooksutamiseks peab esmalt olema arvutis installitud Node-RED-i poolt toetatud Node.js versioon ning seejärel saab selle arvutisse installida. Install võib toimuda mitmel kujul. Esiteks on võimalik sisestada käsureale Node.js käsu, mis installi käivitab. Teiseks võib arvutis Node-RED-i installida arvutiprogrammiga Docker, mis

tegeleb operatsioonisüsteemide virtualiseerimisega, sel viisil jooksutatakse Node-RED-i lõpuks Dockeri ehk virtuaalse operatsioonisüsteemi all. Kolmandaks saab installi läbi viia Linuxil programmiga nimega Snap. Viimaks, kui Node-RED on arvutis installitud, saab selle käivitada sisestades terminalis käsureale alustuskäsu. Node-RED redigeerimiskeskkonnale ligipääsuks tuleb minna brauseriga aadressile „http://localhost" läbi pordi 1880 ehk „http://localhost:1880". (Running Node-RED locally, kuupäev puudub)

Seadmes

Node-RED-i on lisaks arvutile võimalik jooksutada ka osades mikrokontrollerites, mikroarvutites ning Android operatsioonisüsteemil töötavatel telefonides. Mikrokontrolleritest saab Arduino seadmetes jooksutada Node-RED-i, kuid selleks peab Arduinol olema USB ühendus peremees-arvutiga ning samal ajal Node-RED-iga ei saa kasutada programmeerimiskeskkonda Arduino IDE. (Interacting with Arduino, kuupäev puudub) Mikroarvutitest on Raspberry PI ja BeagleBone Boards tooted need, kus võiks samuti jooksutada Node-REDi (Getting Started, kuupäev puudub). Lõputöö tegemise ajal saab kasutada Termux mobiiliäppi Node-RED-i installimiseks ja jooksutamiseks Androidis (Running on Android, kuupäev puudub).

Pilves

Node-RED-i saab paigaldada ka pilve. See on hea variant, kui kasutaja tahab, et programm oleks redigeerimiskeskkonnas koguaeg olemas ja ühesugune ning ei peaks Node.js-i ning Node-RED-i uuesti installima, kui tahta teises arvutis või seadmes seda kasutada. Node-RED-i toetavad pilveteenused on IBM Cloud, Amazon Web Services, Microsoft Azure ja SenseTecnic FRED (Node-RED-i koduleheküg, kuupäev puudub).

Töös kasutab autor Node-RED programmeerimiskeskkonda sellise programmi koostamiseks, mis saadab HTTP päringuid erinevatele REST API-dele ning viib läbi arvutusi ehk hakkab juhtima jahutussõlme tööd.

5.1.1 Alternatiivid Node-RED-ile

Total.js Flow

Total.js Flow on nutistu ehk asjade interneti ning veebirakenduste ühendamiseks mõeldud visuaalse programmeerimise liides, kus programmeeritakse lohista, lase lahti ja ühenda põhimõttel ning mis töötab Total.js raamistikul. Seda on võimalik jooksutada nii enda arvutis kui ka Total.js pilves. (Total.js koduleheküg, kuupäev puudub)

Wia

Wia on pilveplatvorm, mis sarnaselt Node-RED-iga pakub brauseripõhist redigeerimiskeskkonda visuaalseks programmeerimiseks, nimega Flow Studio. (Mishra,

2019) Suurimaks erinevuseks Wia ja Node-RED-i programmeerimiskeskkondade vahel on see, et Flow Studios pole nii palju programmeerimiseks kasutatavaid plokke ning olemasolevad plokid on keskendunud riistvaraga suhtlemisele ning riistvara funktsionaalsuse laiendamisele

Programm seadmes

Visuaalselt programmeerides on mugav luua programmi, sest arendaja ei pea oskama sõlmede sees olevat keelt. Kui aga osata, saab sama programmi kirjutada ka sellise kontrolleri või arvuti sisse, mis interneti ühenduse olemasolul suudaks saata päringuid serveritele ning nendest saadud info põhjal teha arvutusi.

Programmeerimiseks valis autor Node-RED-i, sest sellel on suur kasutajabaas ning foorum olemas kust saab tuge küsida, kui programmeerimisel tekib seisak või viga (Node-RED forum, kuupäev puudub). Teiseks oli Node-RED sobilik valik, sest see sisaldab kogukondliku teeki, kuhu teised kasutajad on üleslaadinud oma valmistatud sõlmi, voolusid ning kollektsioone mõlemast. 07.04.2020 seisuga on selles teegis 2537 sõlme, 1506 voolu ja 117 kollektsiooni (Node-RED Library, kuupäev puudub). Sellest teegist saab vajadusel otsida juba kindlat funktsiooni sisaldavat sõlme või voolu, et mitte ise programmeerimisele aega kulutada. Kõige tähtsamaks põhjuseks miks autor Node-RED-i valis on see, et keskkonnas kasutatakse visuaalset voolupõhist programmeerimist. Autor leidis, et on lihtsam programmeerida kasutades sõlmi ja neid omavahel siduda, kui kirjutada nende asemel koodi, sest tuleb vähem vigu sisse, ei pea programmeerimiskeelt nii hästi oskama ning ei pea nii palju koodiridu kirjutama.

5.2 FRED

Eelnevalt mainis autor, et Node-RED-i on võimalik jooksutada erinevate seadmete peal, sealhulgas ka osades pilveteenustes. Valituks osutus Sensetecnicu pilveteenus nimega FRED ehk *Front End for Node-RED*, mis on spetsiaalselt Node-RED-i võõrustamiseks mõeldud pilv. Pilve kasutamiseks peab aga tellima paketi ja maksma kuutasu ning olenevalt paketist antakse kasutada mingi osa pilve mahust. Olemas on ka tasuta pakett, mis lubab kasutada minimaalset osa pilvest. (Sense Tecnic Systems Inc., kuupäev puudub) Kõigil pakettidel välja arvatud tasuta paketil on võimalik mitmel kasutajal korraga olla sisse logitud ühte peremeeskasutajasse ja töötada samal ajal sama programmi kallal. (Sense Tecnic Systems Inc., kuupäev puudub) Autor tellis pilve kasutamiseks sellise tasulise paketi, millega kaasnevast pilve mahust autori meelest hinnanguliselt piisas töös käsitletava rakenduse programmiliseks lahenduseks loomiseks.

Autor valis Node-RED-i jooksutamiseks FRED pilveteenuse arvuti või muu seadme asemel, sest leidis, et on mugav, kui programm on koguaeg pilves ja ligipääsuks on vajalik vaid internetiühendus. Lisaks ei pea autor ise hoolitsema selle eest, et keskkond oleks töökorras. Teiste pilveteenuste asemel eelistas autor FRED-i, sest Node-RED-i programmilist projekti oli autori arvates palju lihtsam FRED-is üles seada kui teistes pilveteenustes, mis toetavad Node-RED-i.

6 RAKENDUSE KIRJELDUS

Sarnaselt algsele jahutussõlme töökäigule, mida autor kirjeldas peatükis 1, toimub ka peale töös käsitletava rakenduse käivitamist jahutussõlmes vee jahutamine kompressor-jahuti abil, temperatuur milleni vett jahutatakse sõltub etteantud seadepunktist ning vett liigutatakse torudes mööda maja tsirkulatsioonipumba abil. Põhiline erinevus algse ja praeguse töökäigu vahel on see, et nüüd ei tööta tsirkulatsioonipump enam pidevalt ning kompressor-jahuti jahutab vee küll seadepunktini, kuid ainult siis, kui selleks vajadus on, mitte pidevalt. Süsteemi komponentide käivitamine toimub jahutussõlmes peale rakenduse käivitamist vaid juhul, kui tekib jahutusvajadus ja töö peatamine juhul, kui jahutusvajadust pole, näiteks öösel. Lisaks sellele, et komponendid teevad tööd vastavalt vajadusele, jälgib rakendus ka elektribörsi hinda kolm tundi ette ning ajastab komponentide tööd veelgi täpsemalt arvestades tulevasi elektri hindasid, et tööd tehtaks siis, kui on odavam. Need arvestused ning arvestuste jaoks info kogumine viiakse läbi töö programmilises osas ehk rakenduse siseselt.

6.1 Eeldatav jahutusvajadus

Jahutusvajadust käsitletakse selles töös kui muutujat, mis määrab kas jahutussõlme komponendid peavad tööd tegema või mitte ehk kas ruumides olevat õhku on vaja jahutada. Selles töös jaguneb see muutuja kaheks – eeldatavaks jahutusvajaduseks ja hetke jahutusvajaduseks. Eeldatavat jahutusvajadust kasutatakse programmis, et prognoosida kuni kolm tundi ette ruumis oleva temperatuuri tõusu ehk kas peaks ennetatavalt vett paagis jahutama, et siis, kui temperatuuri tõus on toimumas, saaks ära kasutada juba jahutatud vett. Sellise jahutusvajaduse olemasolu arvutatakse programmis, arvestades Päikese otsekiirgusvoogu ehk kui palju energiat langeb Päikeselt ühele ruutmeetrile. Käesolevas töös arvestatakse rakenduses otsekiirgust teiste päikesekiirguse liikide asemel, sest otsekiirgus annab neist kõige enam energiat (Eensaar, 2012). Eeldatavat jahutusvajadust arvutatakse rakenduses uuesti iga tund kolm tundi ette ning kiirgusvoo saamiseks saadetakse samuti iga tund päring Solcast API-le. Solcast on pilveteenus, mis kogub andmeid Maale langeva päikesekiirguse kohta ülemaailma ning teeb ka prognoose (Solcast, 2019). Solcast pilvest otsekiirgusvoo prognoositavate näitude kättesaamiseks saadab rakendus selle pilve API-le päringu. Päringumeetodiks on GET, sest tahame andmeid pilvest lugeda, mitte sinna kirjutada. Päringu URL on https://api.solcast.com.au/world_radiation/forecasts?latitude=58.377983&longitude=26.729038&hours=3&format=json&api_key=********** , milles sisalduvast rajast ning parameetritest saab välja lugeda, et päringu vastuseks

soovitakse saada päikesekiirguse voogude prognoosi kindlate koordinaatide kohta kolm tundi ette ning vastus peab olema JSON formaadis. Koordinaadid viitavad Tartu linnale. Lisaks on päringu parameetrites volitamiseks kasutatud API võtit, mis määrati autorile siis, kui ta Solcast pilve kasutaja tegi. Kuna selle päringu parameetrites pole täpsustatud millist liiki päikesekiirgus voogu vastuseks soovitakse, siis päring tagastab prognoosid kõikide tüüpide kohta, mis pilves saada on. Rakenduses sorditakse andmetest välja otsekiirgusvoo näidud ning need salvestatakse hiljem kasutamiseks. Kuna Solcast API-lt saadud päikesekiirguse andmeid kasutatakse programmi loogikas, mis otsustab kas kompressor-jahutit ning tsirkulatsioonipumpa lülitada sisse või mitte, siis on tähtis, et need andmed oleksid olemas, reaalsed ja õiges formaadis, sest muidu programm ei tööta. Selle kontrollimiseks on rakenduses rida koodi, mis kontrollib et kas pilvest saadud päikesekiirgused näidud on numbrilisel kujul ning kas need jäävad nulli ja 2000 vahele. Kui Solcast pilvest enam andmeid ei anta, antud väärtused pole numbrid või need on normaalvahemikust väljas, siis antakse programmis muutujale *vigaJV* väärtus *true*, mida võetakse arvesse jahutussõlme komponentide lülitamise osas.

6.2 Elektrihindade prognoos

Kui eeldatava jahutusvajaduse põhjal programm ennustab millal on tulevikus vaja jahutada, siis elektrihindade prognoosi kasutatakse programmis jahutamiseks rahaliselt soodsate ajahetkede määramiseks. Tulevaste elektrihindade prognoosi avaldab elektribörs Nord Pool oma turuinfo leheküljel tabeli ja kaardi kujul (Nord Pool Spot AS, kuupäev puudub). Töös käsitletav rakendus pärib sedasama infot aga hoopis Elering Live pilvelt. Elering Live on pilveteenus, mis sisaldab Eesti energiakaubanduse ning elektri- ja gaasisüsteemi toimimise kohta andmeid, mida on võimalik pärida Elering API kaudu (Elering AS, kuupäev puudub). Sarnaselt Solcast pilve API-le tehtud päringule tehakse ka Elering API-le päring kord tunnis ning see peab sisaldama meetodit GET ning URL-i, mis sisaldab õiget rada ja parameetreid, et vastuseks saaks otsitava info. Erinevuseks nende päringute vahel on see, et Elering API ei nõua päringus API võtit, mis tähendab, et päringu võib teha igaüks ja kliendil pole piiratud mitu päringut tal pilve suunas teha on lubatud. Selle päringu URL-is on määratud, et küsitakse pilvest järgmise 24 tunni elektribörsi hindade prognoosi, kuid programmi jahutamise loogika osas arvestatakse vaid esimese kolme tunni hindasid. Kuna Node-RED-i süntaks on JavaScript keeles ning autoril polnud sellega piisavalt kogemusi, siis ei osanud ta teha sellist päringut, kus küsitaks kohe elektribörsi esimese kolme tunni prognoosi 24 tunni prognoosi asemel, nii et dünaamiline kuupäeva muutus päringu sees jääks töötama. Joonis 6.1 on kujutatud funktsioon programmis, mille sisu kasutatakse et saata päring Elering Live API-le.

```
var today = new Date();
var aasta = today.getFullYear();
var kuu = (today.getMonth()+1);
var p2ev = today.getDate();
var tund = today.getHours()+1;

var uusp2ev = today;
uusp2ev.setDate(p2ev+1);
var uusaasta = uusp2ev.getFullYear();
var uuskuu = (uusp2ev.getMonth()+1);
uusp2ev = uusp2ev.getDate();


msg.headers = {};

  msg.url = "https://dashboard.elering.ee/api/nps/price?start=" + aasta + "." + kuu + "." +
  p2ev + "%20" + tund + "%3A00" + "&end=" + uusaasta + "." + uuskuu + "." + uusp2ev + "%20" + tund + "%3A00";

  msg.method = "GET";

return msg;
```

Joonis 6.1 Funktsioon Node-RED keskkonnas, kus programm valmistab ette päringu sisu, et saaks Elering Live API-lt küsida elektribörsi hindasid.

Nagu eeldatav jahutusvajadus on ka elektribörsi API-lt saadud hinnad tähtsad, sest neid kasutatakse programmi osas, mis tegeleb jahutussõlme komponentide lülitamisega. Seega pärast API-lt andmete saamist salvestatakse need programmis ning kontrollitakse, et sisendväärtused oleksid olemas ja õigel kujul. Seda tehakse rakenduses kontrollides kas pilvest saadud elektri hinnad on numbrilisel kujul ning kas need jäävad nulli ja 300 vahele. Kui elektribörsi pilvest enam andmeid ei anta või need on normaalvahemikust väljas, siis antakse programmis muutujale *vigaEl* väärtus *true*, mida võetakse arvesse jahutussõlme komponentide lülitamise osas.

6.3 Voolu muutujad

Elektribörsi hindade ja Päikese otsekiirgus voo kohta andmete kogumiseks vajab rakendus vähest või mitte üldse volitust, et pärida nende pilvedelt infot. Climatix IC ehk pilveteenus, kus hoitakse jahutussõlme komponentide andmepunkte, vajab aga ligipääsuks API võtit ning tunnusvõtit, nagu selgus töö peatükis 4.5, siis kui Climatix IC pilvele saadeti näidispäring. Et rakendus ei vajaks peale sisse lülitamist inimese poolt sekkumist ehk oleks automaatne, siis tuleb programmi siseselt tunnusvõtit pidevalt uuendada, kuna tunnusvõti ei jää igaveseks kehtima. Seda uuendatakse saates Climatix IC API-le päringumeetodiga POST, kus kirjutatakse pilve Climatix IC kasutajanimi ja parool ning kui kauaks tunnusvõti kehtima peaks – selles päringus on määratud, et tunnusvõti peaks kehtima maksimaalselt lubatud aeg ehk 20160 minutit ehk kaks nädalat (Joonis 6.2). Päringu vastuse keha sisaldab tunnusvõtit, mida kasutatakse programmis edaspidi autoriseerimiseks kõikides päringutes, mis saadetakse Climatix IC

API-le. Seda ja sarnaseid muutujaid kutsutakse programmi erinevates sõlmedes välja *flow.get()* käsuga.

```
var key = flow.get('clientRTAkey');
var email = flow.get('clientRTAemail');
var parool = flow.get('clientRTAparool');

msg.headers = {};
msg.method = "POST";
msg.url = "https://api.climatixic.com/Token/";
msg.headers['Ocp-Apim-Subscription-Key'] = key;
msg.payload = 'grant_type=password&username='+email+'&password='+parool+'&expire_minutes=20160';
return msg;
```

Joonis 6.2 Päringu sisu Climatix IC pilvest tunnusvõtme küsimiseks

6.4 Hetke jahutusvajadus

Kui eeldatavat jahutustvajadust kasutatakse rakenduses, et ennetatavalt jahutussõlmes vett jahutada tuleviku tarbeks, siis hetke jahutusvajaduse järgi otsustab programmiloogika kas olevikus on jahutussõlmes vett jahutada ruumide jahutamise tarbeks. See muutuja on programmi toodud selleks, et kui prognoositavate andmete põhjal võib programm teha vale otsuse, siis selle kasutamisega kindlustatakse, et ruum ei jää jahutamata. Selle muutuja arvutamiseks on büroohoone ühele korrusele pandud erinevates ilmakaartes asuvatesse ruumidesse kokku viis Siemensi RDS120 nutitermostaati. Need termostaadid mõõdavad ruumides temperatuuri ning läbi termostaadi puutetundliku kuvari kasutajate poolt sisestatud seadepunktide käib ka ruumides temperatuuri juhtimine. See tähendab, et kui ruumides on külm, siis köetakse õhku radiaatoriga, ja kui on kuum, siis jahutatakse vesijahutus konvektoriga. Need nutitermostaadid on ühendatud Climatix IC pilvega ja seetõttu on võimalik töös käsitletaval rakendusel küsida sellest pilvest termostaatide poolt kogutud andmeid. Hetke jahutusvajaduse arvutamiseks küsib rakendus pilvest iga tund termostaatide kõige värskemad temperatuuri näidud ning seadepunktid, mis pilves saadaval on. Rakendus kasutab Climatix IC API-le saadetavas päringus selle autoriseerimiseks varem pilvest küsitud tunnusvõtit ning pärib igalt nutitermostaadilt kõik andmepunktid, millest programm sordib järgmises funktsioonis välja ruumi temperatuuri ning seadepunkti. Climatix IC API-le saadetava päringu sisu illustreerib Joonis 6.3.

```
var APIv6ti = flow.get('clientRTAkey');
var tunnusv6ti = flow.get('clientRTAtkn');

msg.headers = {};
  msg.headers['Ocp-Apim-Subscription-Key'] = APIv6ti;
  msg.headers['Authorization'] = tunnusv6ti;
  msg.url = "https://api.climatixic.com/DataPoints/Values?parentId=[{"+'"'+
  "Plants"+'"'+": "+'"'+"P8dd20ec9-6a6d-4407-939c-0eb67062d57f"+'"'+"}]";
  msg.method = "GET";
return msg;
```

Joonis 6.3 Päringu sisu Climatix IC pilve API-le ühe nutitermostaadi andmepunktide küsimiseks

Peale sortimist võrdleb programm iga ruumi temperatuuri ja seadepunkti paari eraldi. Kui ruumi temperatuur on kõrgem kui seadepunkt + 2 kraadi, siis järelikult on vaja ruumi jahutada ehk ruumil on jahutusvajadus. Seadepunkti väärtusele lisatakse programmis 2, sest ka ruumi jahutuse seadepunkt on termostaadi seadepunkt +2, see tähendab, et vesijahutus-konvektor hakkab ruumi jahedat õhku puhuma sellel kraadil ja sellepärast sobib ka see rakenduses näitamaks hetke jahutusvajaduse olemasolu. Kui vähemalt kahel ruumil viiest on jahutusvajadus, siis saadetakse jahutamise loogika programmi osasse väärtus, mis määrab, et on olemas hetke jahutusvajadus ehk jahutussõlmes on kohe vaja vett jahutada. Programm peab tuvastama, et vähemalt kahel ruumil viiest on jahutusvajadus, sest see muudab vähem tõenäolisemaks juhtumi, kus programm teeb vale otsuse juhul, kui mitmes ruumis korraga peaks toimuma lühiajaline temperatuuri langus või kui mõnes ruumis on ruumikasutaja seadnud kõrge seadepunkti. Sel juhul piisab, kui alla pooltel ruumidel on jahutusvajadus, et jahutussõlm tööd teeks, sest autor peab ruumikasutaja mugavust tähtsamaks kui kokkuhoidu energia või raha arvelt. Hetke jahutusvajaduse arvutust kujutab Joonis 6.4.

```
var temp1 = flow.get('temp1');
var temp2 = flow.get('temp2');
var temp3 = flow.get('temp3');
var temp4 = flow.get('temp4');
var temp5 = flow.get('temp5');
var SP1 = flow.get('SP1');
var SP2 = flow.get('SP2');
var SP3 = flow.get('SP3');
var SP4 = flow.get('SP4');
var SP5 = flow.get('SP5');

var loendur=0;
var hetkeJV;
var viga = false;

if (temp1>SP1+2) loendur++;
if (temp1<10 || temp1>30 || isNaN(temp1)) viga = true;

if (temp2>SP2+2) loendur++;
if (temp2<10 || temp2>30 || isNaN(temp2)) viga = true;

if (temp3>SP3+2) loendur++;
if (temp3<10 || temp3>30 || isNaN(temp3)) viga = true;

if (temp4>SP4+2) loendur++;
if (temp4<10 || temp4>30 || isNaN(temp4)) viga = true;

if (temp5>SP5+2) loendur++;
if (temp5<10 || temp5>30 || isNaN(temp5)) viga = true;

if (loendur>=2) hetkeJV=true;
else hetkeJV=false;

flow.set('vigatemp',viga);
flow.set('hetkeJV',hetkeJV)

return msg;
```

Joonis 6.4 Hetke jahutusvajaduse arvutamine

Ka hetke jahutusvajaduse arvutamisel tehakse ruumi temperatuuri väärtustele programmis kontroll juhuks, kui mõnel ruumis oleval termostaadil esineb rike ja ei näita enam reaalsed väärtusi. Rakenduses kontrollitakse kas ruumi temperatuur on numbrilisel kujul ja kas jääb 10 ja 30 vahele. Kui kontroll tuvastab, et vähemalt üks termostaat ei näita õigeid andmeid, siis antakse programmis muutujale *vigatemp* väärtus *true*, mida võetakse hiljem arvesse. Enne, kui rakendus läheb edasi jahutamise loogika funktsiooni juurde, pärib see Climatix IC pilvest ka jahutusõlme pealevoolu vee temperatuuri. Pealevoolu vee all mõeldakse töös vett, mida pumbatakse paagist ruumidesse.

6.5 Jahutamise loogika

Selles programmi osas määratakse ära millistel tingimustel rakendus lülitab sisse kompressor-jahuti ning tsirkulatsioonipumba ja tehakse lülitamise osas otsus. Esmalt kontrollib programm kas hetkel on tööpäev või nädalavahetus. Kui tegemist on tööpäevaga, siis programm hakkab võrdlema varasemalt kogutuid andmeid ning otsustab kas lülitada jahutussõlm sellisesse töörežiimi, kus kompressor-jahuti ning tsirkulatsioonipump käivituvad. Kui on aga nädalavahetus, siis jahutussõlm lülitatakse töörežiimi, kus kompressor-jahuti ja tsirkulatsioonipump seiskuvad. See kontroll on mõeldud selleks, et nädalavahetusel need jahutussõlme komponendid seisaks, sest ruume pole mõtet jahutada, kui inimesed neid ei kasuta. Sama põhimõtte pärast on programmis ka teine kontroll, mis kontrollib päevade asemel kellaaegu. Programm määrab, et kui kell on 7:00 kuni 19:00, siis vaadatakse rakenduses järgmisi kontrolltingimusi, kui kell on aga vähem kui 7:00 või rohkem kui 19:00, siis jahutussõlme komponente sisse ei lülitata. Töös käsitletaval jahutussõlme kontrollerile on programmeeritud kolm töörežiimi. Esimene neist on automaatrežiim, mille järgi töötas jahutussõlm enne töös käsitletava rakenduse kasutamist ehk tsirkulatsioonipump töötab pidevalt ning kompressor-jahuti jahutab vett seadepunktini. Sellesse režiimi saab jahutussõlme panna, kui saata API-ga töörežiimi andmepunktile väärtus 0. Kui jahutussõlm on teises režiimis, siis kompressor-jahuti seisab ning tsirkulatsioonipump töötab pidevalt. Teist režiimi rakenduses ei kasutata. Kolmandas režiimis on mõlemad jahutussõlme komponendid seiskunud ning sellesse režiimi saab jahutussõlme panna, kui saata API-ga töörežiimi andmepunktile väärtus 2. Pärast teist kontrolli võrdleb programm, et kas päikesekiirguse prognoosiga saadud andmete kohaselt on kolme tunni pärast Päikese otsekiirguse voog suurem kui 350 W/m^2. Piiriks võeti 350 W/m^2, sest katse käigus selgus, et rakendus töötab kõige efektiivsemalt, kui arvestame, et eeldatav jahutusvajadus tekib juba siis, kui tuvastatakse Päikese otsekiirgus vähemalt 350 W/m^2. Sellel otsekiirguse väärtusel kasvas kuni ühe tunni jooksul kahes päikese poolses ruumis temperatuur ruumi jahutuse seadepunktini. Lisaks kontrollib programm kas elektribörsi hind praegusel tunnil on parem kui ühe, kahe ja kolme tunni pärast. Kui need kaks tingimust vastavad tõele, siis antakse muutujale *olek* programmis väärtus 0, mille peale hiljem käivitatakse kompressor-jahuti ning tsirkulatsioonipump, et vett jahutada kolm tundi ette, sest on teada, et kolme tunni pärast on jahutust vaja ning hetkel on nendest kolmest tunnist kõige parem hind. Kui aga prognoosi kohaselt kolme tunni pärast Päikese otsekiirguse voog ei ületa 350W/m^2 ehk eeldatavat jahutusvajadust ei ole või elektri hind on praegusel tunnil kõrgem kui järgmistel, siis liigub programm edasi järgmiseid tingimusi kontrollima. Järgmine tingimuste kontroll on peaaegu sama nagu eelmine ehk kontrollitakse kas kahe tunni pärast on eeldatav jahutusvajadust ning kas

praegune elektribörsi hind on madalam kui ühe ja kahe tunni pärast. Kui tingimused on tõesed, siis antakse muutujale *olek* väärtus 0. Kui aga ka kahe tunni pärast pole jahutusvajadust või kui elektri hind pole tollel tunnil madalam kui järgmistel tundidel, siis viib programm läbi järgmise kontrolli. Kolmandas tingimuste kontrollis kontrollitakse eeldatava jahutusvajaduse olemasolu järgmisel tunnil ning kas selle tunni elektribörsi hind on parem kui järgmisel. Lisaks kontrollitakse kas veepaagist väljuva vee temperatuur on kõrgem kui 18 °C . Kui programm tuvastab, et järgmisel tunnil on eeldatav jahutusvajadus ning kas praegune elektribörsi hind on madalam kui järgmisel tunnil või vee temperatuur on kõrgem kui 18 °C, siis antakse muutujale *olek* väärtus 0. Programm kontrollib veepaagist väljuva vee temperatuuri, sest kui vett pole mõnda aega jahutatud ja järgmine tund on vaja jahutatud vett kasutada, siis ei lase programm tõusta pealevoolu vee temperatuuril liiga kõrgele ehk mitte kõrgemale kui 18 °C, et kompressor-jahuti jõuaks alati õigeks ajaks vee jahutada seadepunktile vastavale temperatuurile, milleks selle töö tegemise ajal oli 13 °C. Viimane tingimus, mida selles programmi osas kontrollitakse on see, et kas on olemas hetke jahutusvajadus. Hetke jahutusvajadus arvutati eelnevalt kontrollides ruumide temperatuure ning seadepunkte. Kui programm tuvastab, et jahutusvajadus on olemas, siis antakse muutujale *olek* väärtus 0 ehk järgmises programmi osas käivitatakse kompressor-jahuti ning tsirkulatsioonipump, et ruume kohe jahutama hakata. See viimane kontroll kindlustab, et ruumid ei jää kunagi jahutamata. Kui mitte ükski neist tingimustest ei olnud täidetud, siis jahutussõlme komponente sisse ei lülitata ning muutuja *olek* väärtus on 2. Seda funktsiooni kujutab Joonis 6.5.

```
var JV = flow.get('JVarray');
var hetkeJV = flow.get ('hetkeJV');
var hind = flow.get('array');
var flow = flow.get ('flowtemp');
var olek = 2;
var today = new Date();
var tund = today.getHours();
var n = today.getDay(); // 0 - pühapäev, 6 - laupäev

if(n>0 && n<6){ //kui on tööpäev siis
    if (tund>4 && tund <16){
        if (JV[5]>350 && hind[0]<hind[1] && hind[0]<hind[2]&& hind[0]<hind[3] ){
            olek = 0;
        }

        else if (JV[3]>350 && hind[0]<hind[1] && hind[0]<hind[2] ){
           olek = 0;
        }

        else if (JV[1]>350 && (hind[0]<hind[1] || flow>18) ){
           olek = 0;
        }

        else if (hetkeJV===true){
            olek = 0;
        }
        else olek = 2;
    }
    else olek = 2;
}
else olek = 2;

flow.set('olek',olek);

return msg;
```

Joonis 6.5 Funktsioon, kus programm otsustab millisesse töörežiimi jahutussõlm pannakse

6.6 Väljund

Kui rakenduses on ära otsustatud kas komponente on tarvis sisse lülitada või mitte, siis programmi järgmine samm on liikuda funktsiooni sisse, mis sisaldab vajalikku infot, et saata päring Climatix IC API-le. Selle päringu sisu on muuta jahutussõlme kontrolleri töörežiimi andmepunkti väärtuseks 0 ehk käivitada tsirkulatsioonipump ja kompressor-jahuti või muuta see töörežiimi väärtuseks 2 ehk need komponendid seisata. Funktsioonis küsitakse käsuga *flow.get*() eelmistest programmi osadest muutujate *olek*, *vigaJV*, *vigaEl* ning *vigatemp* väärtused. Viimase kolme muutuja väärtused näitavad kas eeldatava jahutusvajaduse, elektribörsi hindade või hetke jahutusvajaduse väärtuste arvutamisel või pärimisel programm tuvastas, et need väärtused ei olnud numbrid või ei olnud normaalses vahemikus. Kui vähemalt ühel nendest kolmest muutujast on väärtus

true, siis pannakse jahutussõlm automaatsesse režiimi ehk pidevalt töötavasse režiimi nagu see enne töös käsitletava rakenduse käivitamist oli. Seda sellepärast, et isegi kui rakendus ei tööta korrektselt peavad ruumid saama jahutatud. Kui ühtegi viga aga ei tuvastata, siis olenevalt eelmises funktsioonis muutujale *olek* antud väärtusest annab rakendus jahutussõlmele uue režiimi saates Climatix IC API-le päring kasutades PUT meetodit andmete uuendamiseks. Kui peatükis 4.2 ütles autor, et PUT päringumeetodit kasutades uuendatakse kogu ressursi sisuga sellega, mis kehas kirjeldatud on, siis Climatix IC API võimaldab PUT meetodit kasutades uuendada ka vaid ühte osa ressursist, milleks praegusel juhul on muutuja *value* (Climatix IC Cloud API, API Introduction). Päringu saatmise tulemusena uuendatakse pilves jahutussõlme töörežiimi andmepunkti ning pilv saadab uuendatud andmed edasi kontrollerisse, kus siis uuendatakse samuti töörežiimi ning vastavalt sellele seisatakse või käivitatakse komponendid. Seda osa programmist kujutab Joonis 6.6.

```
1  var key = flow.get('clientRTAkey');
2  var token = flow.get('clientRTAtkn');
3  var olek = flow.get('olek');
4  var vigaJV = flow.get('vigaJV');
5  var vigaEl = flow.get('vigaEl');
6  var vigatemp = flow.get('vigatemp');
7  if(vigaJV || vigaEl || vigatemp) olek=0;
8
9
10 msg.headers = {};
11   msg.url = "https://api.climatixic.com/Pfb48db52-e198-"+
12   "450f-9d96-b255b68743f4;1!6WHH75YJKQWBE2";
13   msg.method = "PUT";
14   msg.headers['Ocp-Apim-Subscription-Key'] = key;
15   msg.headers['Authorization'] = token;
16 if (olek === 0){
17     msg.payload = {"value": 0};
18 }
19 else if (olek === 2){
20     msg.payload = {"value": 2};
21 }
22
23   return msg;
```

Joonis 6.6 Funktsioon, kus programm valmistab päringu sisu ette Climatix IC pilve saatmiseks

Algselt plaanis autor lisada programmi funktsiooni, mis oleks saatnud meili rakenduses paika pandud aadressile juhul, kui sisendandmete kogumisel või nende kasutamisel tuvastati viga ja selle läbi jahutussõlm automaat režiimi pandi. See funktsionaalsus Node-RED-ist kaotati käesoleva töö valmistamise ajal autorile teadmata põhjustel. Siiski vea tuvastamisel rakendus ei seisku ehk kontrollib iga tund vigade olemasolu ning vigade kadumisel suudab programm oma tavapärast töökäiku jätkata.

7 VÕRDLUS

7.1 Rakenduse funktsionaalsuse kontroll

Peale rakenduse käivitamist viis autor läbi kontrolli, et välja selgitada kas jahutus ruumides toimib efektiivselt ehk kas ruumides paiknevad vesijahutus-konvektorid suudavad ruume ära jahutada, kui jahutussõlmest konvektorite soojusvahetitesse voolav vesi on jahutatud rakenduse programmi järgi. Kontrollimiseks päris autor Climatix IC pilvest nutitermostaatide temperatuuride ja seadepunktide ajalood ühe päeva kohta. Andmepunktide ajaloo päringu vastus sisaldab andmepunkti väärtust ning ajahetke, mil väärtus pilve salvestati. Ajahetk sõltub intervallist, mille klient päringu päises täpsustab. Selle töö raames määrati päringus intervalliks üks tund. Kontrolli põhimõtteks oli võrrelda iga ruumi temperatuuri ja seadepunkti. Kui andmeid võrreldes on näha, et temperatuur tõusis üle jahutuseseadepunkti, järsult langes ja jäi ruumi seadepunkti ning jahutuse seadepunkti vahele, siis võib järeldada, et rakenduse järgi töötav jahutussõlm jahutas vett ruumi jahutamise tarbeks piisavalt hästi. Kogutuid andmeid kontrollides selgus, et ainult ühes ruumis tõusis sel ööpäeval ruumi temperatuur üle seadepunkti. Selles ruumis olevat nutitermostaati kutsub autor selles töös nutitermostaat nr.1-ks. Kui vaadata Joonis *7.1* kujutatud nutitermostaat nr.1 andmeid, siis on märgata, et temperatuur püsis enamasti ennenähtud vahemikus, kuid kell 11:00 ületas see jahutuse seadepunkti. Kell 12:00 on aga temperatuur langenud ühe kraadi võrra, mis tuleneb ilmselt sellest, et vesijahutus-konvektor käivitus enne kella 12:00 ja jahutas ruumi. Kuna temperatuuri langetati edukalt, siis järeldab autor sellest, et rakenduse järgi töötav jahutussõlm tõepoolest jahutas vee ruumi jahutamise tarbeks piisavalt hästi. Kahjuks ei salvestata jahutussõlme töörežiimi, tsirkulatsioonipumba ega ka kompressor-jahuti töökäsu andmepunkte Climatix IC ajalukku, muidu oleks saanud autor tagantjärgi kontrollida kas sel päeval jahutussõlme komponente käivitati.

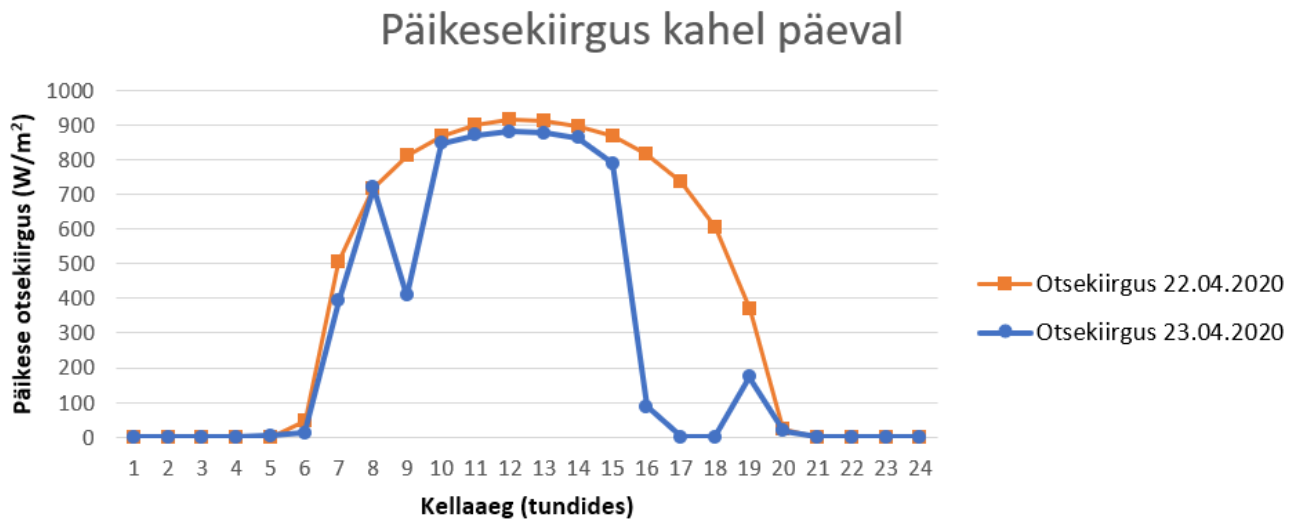
	Nutitermostaat 1		
Kellaaeg	**Seadepunkt**	**Temperatuur**	**Jahutuse seadepunkt**
01:00	21.0	20.6	23.0
02:00	21.0	20.6	23.0
03:00	21.0	20.7	23.0
04:00	21.0	20.9	23.0
05:00	21.0	21.2	23.0
06:00	21.0	21.3	23.0
07:00	21.0	21.9	23.0
08:00	21.0	22.0	23.0
09:00	21.0	22.8	23.0
10:00	21.0	23.0	23.0
11:00	21.0	23.3	23.0
12:00	21.0	22.3	23.0
13:00	21.0	22.1	23.0
14:00	21.0	22.1	23.0
15:00	21.0	22.0	23.0
16:00	21.0	21.7	23.0
17:00	21.0	21.5	23.0
18:00	21.0	21.3	23.0
19:00	21.0	20.9	23.0
20:00	21.0	20.8	23.0
21:00	21.0	20.6	23.0
22:00	21.0	20.5	23.0
23:00	21.0	20.5	23.0
00:00	21.0	20.5	23.0

Joonis 7.1 Nutitermostaat nr.1 ühe ööpäeva andmed

7.2 Energiatarve

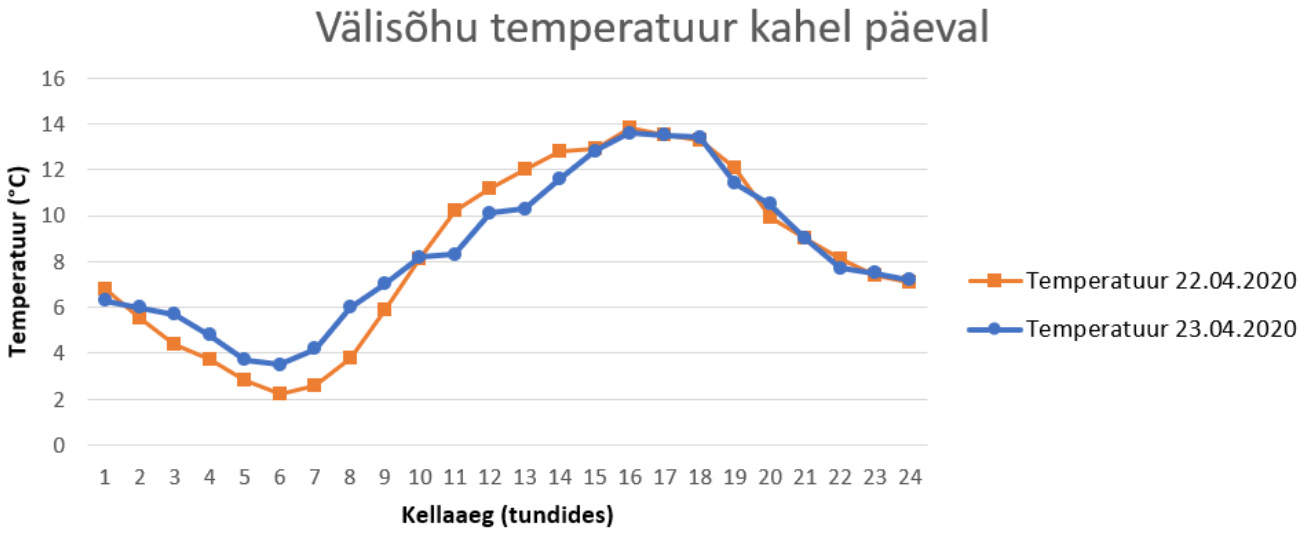
Et välja selgitada kui palju või kas üldse langeb energiatarve rakenduse käivitamisel jahutussõlme komponentide vajaduspõhise töötamise arvelt, mõõtis autor kahel sarnase langeva Päikese otsekiirguse voo ja välisõhu temperatuuriga ööpäeval energiatarvet. Ühel neist päevadest töötas jahutussõlm automaat töörežiimis nagu varem ning teisel päeval töötas jahutussõlm rakenduse programmiloogika järgi. Katsete päevadel pidid otsekiirguse vood ja välisõhu temperatuurid olema sarnased, sest Päikese otsekiirgus mõjutab ruumi temperatuure ning välisõhu temperatuurist sõltub kui palju energiat peab kompressor-jahuti kulutama, et vett jahutada. Seega, mida sarnasemad päevad, seda täpsema energiatarvete erinevuse katsete põhjal saab. Esimene katse viidi läbi kuupäeval 22.04.2020. Selle ööpäeva päikesekiirguse kõrgeim näit oli 915 W/m^2, mis salvestati kell 12.00 ning ööpäeva kõrgeim temperatuuri näit oli 13,8 °C, mis salvestati

kell 16:00. Teine katse viidi läbi järgmisel päeval ehk kuupäeval 23.04.2020. Selle ööpäeva päikesekiirguse kõrgeim näit oli 881 W/m^2, mis salvestati kell 12.00 ning ööpäeva kõrgeim temperatuuri näit oli 13,6 °C, mis salvestati kell 16:00. Joonis 7.2 kujutab graafikus mõlema päeva Päikese otsekiirgusvoogude näite, kus näidud on võetud iga tunni tagant. Sellelt graafikult on näha, et päikesekiirgused pole identsed, kuid on autor arvates piisavalt sarnased, et neid kahte ööpäeva omavahel võrrelda.



Joonis 7.2 Päikesekiirgus mõlemal ööpäeval

Joonis 7.3 kujutab graafikus mõlema ööpäeva välisõhu temperatuuri näite, kus näidud on võetud iga tunni tagant. Sellelt graafikult on näha, et ka temperatuurid pole neil kahel ööpäeval identsed, aga paari kraadine temperatuuride erinevus ei mõjuta autori arvates kompressori energiatarvet piisavalt, et neid kahte ööpäeva ei saaks võrrelda.



Joonis 7.3 Välisõhu temperatuur kahel ööpäeval

Päikese otsekiirguse näidud nende katse jaoks võeti Solcast pilvest ning temperatuuri näidud Tartu Ülikooli füüsika instituudi e-ilmajaama andmebaasist (Meteo, Temperatuur). Eelmainitud kuupäevadel mõõtis autor jahutussõlme pandud energiaarvestite abil kompressor-jahuti ning tsirkulatsioonipumba energiatarvet. Kasutatud arvestid võimaldasid autoril salvestada iga tund uue energiatarbe väärtuse, et tunnis kulutatud energia väärtust saaks hiljem kogu energiatarbe ja rahalise kulu arvutustel kasutada. Kuupäeval 22.04.2020, ehk päeval mil rakendust ei olnud käivitatud, mõõtsid arvestid komponentidelt kokku 308,71 kWh energiakulu. Kuna jahutussõlm oli sellel ööpäeval automaat töörežiimis, siis töötas tsirkulatsioonipump kõik 24 tundi järjest ning kompressor-jahuti jahutas ööpäevaringselt sõlmes vett 13 kraadini. Autor tegi katse tarbeks rakenduse programmi sellise ajutise muudatuse mille tõttu rakendus käivitus kell 00:01 kuupäeval 23.04.2020. Sel päeval mõõtsid arvestid komponentidelt kokku 122,336 kWh. Võrreldes nende kahe päeva energiatarvet selgub, et teisel päeval kulutasid kompressor-jahuti ning tsirkulatsioonipump kokku 186,374 kWh võrra vähem energiat. See tähendab, et teisel päeval kulutati ümardatult 2,5 korda vähem energiat kui esimesel päeval. Arvestid salvestasid iga tund lisaks uuele energiatarbe näidule ka ajajälje, mille järgi oli võimalik välja selgitada millisel tunnil kui palju energiat kulus. Kuna arvestitelt on teada iga tunni energiatarve ja Nord Pool elektribörsi andmebaasist iga tunni elektri hind, siis sai autor täpselt välja arvutada kui suure rahalise kulu tekitas mõlemal päeval jahutussõlme komponentide töö. Esimesel päeval kulutatud 308,71 kWh energiat läks maksma 10,64 € ning teisel päeval kulutatud 122,336 kWh läks maksma 3,87 € ehk ümardatult 2,75 korda ehk 6,77 € võrra vähem kui esimesel päeval. Tasub märkida, et ööpäeva keskmine ühe MWh hind esimesel päeval oli 34,42 € ning teisel päeval 32,76 € ehk 1,66 € võrra väiksem. Neid hindu arvestades oleks esimesel päeval kulutatud energia läinud maksma 10,62 € ja teisel päeval 4,01 € ehk nende vahe oleks olnud ümardatult 2,65 kordne. Seetõttu arvab autor, et nii vähene ööpäevaste keskmiste hindade erinevus ei avalda lõpptulemusele suurt mõju ehk võib öelda, et selles katses tulenes rahalise kulu alandus rakenduse tööst ning mitte sellest, et ööpäevadel oleksid tulemust muutvalt erinevad elektri hinnad olnud.

KOKKUVÕTE

Lõputööle seadis autor mitu eesmärki, millest osa keskendus jahutusõlme juhtiva rakenduse ehituse määramisele ning teine osa lugejale rakenduses kasutavate tööpõhimõtete ja tarkvarade selgitamisele. Esiteks kirjeldas autor kuidas toimis jahutussõlm ning büroohoone ruumide kliimatehnika enne, kui ta oli käivitanud lõputöö käigus loodud rakenduse ning selgitas, et pannes jahutussõlme kontrolleri ning ruumide kliimatehnikat juhtivad nutitermostaadid Climatix IC pilve saab hiljem neid andmeid Internetist välja lugeda. Et lugeja mõistaks kuidas on rakendusel võimalik pärida erinevatelt pilveteenustelt infot selgitas autor mis on API ja mis tüüpe API-sid on olemas, lõpuks peatudes REST tüüpi API-l. Kuna REST API kasutamisoskus oli selle rakenduse valmistamisel väga tähtis osa, siis kirjeldati töös kuidas luua API-le päringuid ning mida päring peaks sisaldama, et vastuseks saaks päringu saatja ehk klient soovitud info või et päring andmete uuendamiseks pilves läbi läheks.

Kuna keskkond, kus rakenduse programmeerimine ning jooksutamine töös läbi viidi, võib autori arvates olla lugejale tundmatu, siis tutvustas autor lühidalt Node-RED-i ning ka seda sisaldavat pilve nimega FRED. Autor kasutas Node-RED-i visuaalse programmeerimise keskkonda eesmärgil, et tal oleks lihtsam programmeerida JavaScript keeles.

Järgmiseks selgitas autor kuidas peale rakenduse käivitamist jahutussõlme töökäik hakkab erinema varasemast. Erinevus kahe töökäigu vahel on see, et rakenduse programmi osa prognoosib esmalt kuni kolm tundi ette kas on tekkimas ruumides jahutusvajadus ning siis jahutab jahutussõlmes asuvas veepaagis vett, kui elektribörsi hind on tollel tunnil odavam kui lähitulevikus. Lisaks ei tee peale rakenduse käivitamist jahutussõlm tööd, kui selleks pole vajadust, nagu näiteks öösel. Autor tutvustas lugejale programmi sisu ning tingimusi, mis käivitavad või peatavad jahutussõlme komponentide töö, et lugeja saaks mõista kuidas üritab rakendus säästa energiat ja seeläbi ka raha.

Et kontrollida kas valminud rakendus töötab eesmärgipäraselt ning kas või kui palju väiksem on energiakulu rakendust kasutades viis autor läbi mitu kontrollkatset. Esimene neist kontrollis kas rakenduse järgi töötav jahutussõlm jahutab ruumidesse voolavat vett piisavalt, et vajaduse tekkimisel vesijahutus-konvektor suudaks ruumi ära jahutada. Selle katse põhjal selgus, et konvektoril oli edukalt võimalik ruum ära jahutada. Ülejäänud kahel katsel mõõdeti kompressor-jahuti ja tsirkulatsioonipumba energiakulu kahel kuupäeva poolest erineval, kuid sarnase temperatuuri ja Päikese otsekiirgusega päeval. Katsete tulemusel selgus, et ööpäeval, mil jahutussõlme komponentide juhtimiseks kasutati rakendust, oli kompressor-jahuti ja tsirkulatsioonipumba kogu energiakulu umbes 2,5 korda väiksem ehk kulutati 186,374 kWh võrra vähem energiat.

Rahalise kulu erinevus energiatarbe arvelt oli neil kahel päeval umbes 2,75 kordne ehk päeval mil kasutati rakendust jahutussõlme komponentide juhtimiseks kulutati 6,77 € võrra vähem elektrienergia peale kui päeval mil rakendust ei kasutatud.

Autori arvates õnnestus sellise rakenduse loomine, mis täidab peatükis 1.1 ja sissejuhatuses kirjeldatud eesmärke ehk rakendus ajastab jahutussõlme komponentide tööd programmis ettenähtud tingimustele, sealhulgas arvestades elektribörsi hinda ning hetke ja eeldatavat jahutusvajadust. Samuti suudab rakendus jahutussõlme tööd mõjutades alandada energia ja rahalist kulu samal ajal säilitades ruumi kasutaja mugavus. Autor leiab, et energia ning rahaline sääst rakendust kasutades on piisav, et seda ka edaspidi selles jahutussõlmes kasutama jääda, kuid saab aru, et rakendust on veel võimalik täiendada. Autor usub, et välisõhu temperatuuri kasvades tõuseb ka rakenduse kasulikkus kulude alandamisel.

SUMMARY

The author set multiple objectives for this thesis. One part of the objectives focused on determining the structure of the application that will optimise the operation of a cooling plant and making the application while the other part focused on explaining the working principles and software used for making the application. Firstly the author described how the cooling plant and the technology controlling room climate had worked before he activated the application made during the thesis and explained that by connecting the controller of the cooling plant and the smart thermostats controlling the room climate technology with Climatix IC cloud, information can later be read out from the devices over the Internet. In order for the reader to understand how the application can request information from different cloud services, the author explained what an API is and what types of APIs exist, finally stopping at REST API. As the ability to use the REST API was a very important part in the development of this application, the author described how to create requests to the API and what the request should contain so that the sender receives the proper information or the request to update data in the cloud would go through.

Since the environment in which the author carried out the programming and running of the application may be unknown to the reader, the author briefly introduced Node-RED and the cloud service containing it called FRED. The author used the Node-RED visual programming environment in the purpose to make it easier for him to program in JavaScript.

Next, the author explained how the operation of the cooling plant will differ from the previous one after starting the application. The difference between the two operations is that program part of the application will first forecast up to three hours in advance whether a cooling demand will be occurring in the rooms of the bureau building and will then cool the water in the cooling plant's water tank if the power exchange price is cheaper at that hour than in the near future. In addition, after activating the application, the cooling plant will not work when it is not needed, for example at night. The author introduced the content of the program and the conditions that start or stop the operation of the cooling plant components to the reader, so that the reader could understand how the application tries to save energy and by that also money.

In order to check whether the completed application works as intended and whether or how much lower is the energy consumption using the application, the author conducted several control tests. The first of these checked whether the cooling plant operating using the application cools the water flowing into the rooms sufficiently so that, if

necessary, the fan coil unit could cool the room. This test showed that the fan coil unit was able to cool the room successfully. In the other two tests, the energy consumption of the compressor-chiller and the circulation pump was measured on two different dates with similar temperatures and direct solar radiation. The tests showed that on the day when the application was used to control the components of the cooling plant, the total energy consumption of the compressor-chiller and the circulation pump was about 2,5 times lower i.e. 186,374 kWh less energy was consumed. The difference in financial cost due to energy consumption was about 2,75 times on those two days, i.e. on the day that the application was used to control the components of the cooling plant 6,77 € less was spent on electricity than on the day that the application was not used.

In the opinion of the author, the making of an application that fulfills the objectives described in chapter 1.1 and in the introduction was a success, i.e. the application schedules the operation of the cooling plant components under the conditions prescribed in the program, including taking in account the power exchange price and current and expected cooling demand. The application is also able to reduce energy and financial costs while maintaining the comfort of the room user by influencing the operation of the cooling plant. The author finds that the energy and financial savings using the application are sufficient to continue to use it in this cooling plant but understands that it is still possible to improve the application. The author believes that as the outdoor temperature increases so shall increase the utility of the application in reducing costs.

KASUTATUD ALLIKATE LOETELU

About. (kuupäev puudub). Kasutamise kuupäev: 7. aprill 2020. a., allikas Node-RED: https://nodered.org/about/

Arhiiv. (11. märts 2017. a.). Kasutamise kuupäev: 2. mai 2020. a., allikas Meteo: https://meteo.physic.ut.ee/

Barker, E. (2016). *Recommendation for Key Management - Part 1 General (Revision 4)*. Gaithersburg, Maryland, USA: National Institue of Standards and Technology.

Cloudflare. (kuupäev puudub). *What Do Client-Side and Server-Side Mean? | Client Side vs. Server Side*. Kasutamise kuupäev: 16. märts 2020. a., allikas Cloudflare: https://www.cloudflare.com/learning/serverless/glossary/client-side-vs-server-side/

Davis, T. (23. detsember 2019. a.). *What is An API and How Does It Work?* Kasutamise kuupäev: 16. märts 2020. a., allikas Towards Data Science: https://towardsdatascience.com/what-is-an-api-and-how-does-it-work-1dccd7a8219e

Eensaar, A. (2012). *Päikesekiirgus atmosfääris.* Õpiobjekt, Tallinna Tehnikakõrgkool, Arhitektuuri ja keskkonnatehnika teaduskond. Kasutamise kuupäev: 1. mai 2020. a., allikas http://eprints.tktk.ee/128/2/paikesekiirgus/kiirguskliima_iserasused.html

Elering AS. (kuupäev puudub). *Elering LIVE*. Kasutamise kuupäev: 27. aprill 2020. a., allikas Elering LIVE: https://dashboard.elering.ee/

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures.* Doktoriväitekiri, California Ülikool, Irvine.

Getting Started. (kuupäev puudub). Kasutamise kuupäev: 24. märts 2020. a., allikas Node-RED: https://nodered.org/docs/getting-started/

Gilling, D. (17. veebruar 2017. a.). *Steps to Building Authentication and Authorization for RESTful APIs*. Kasutamise kuupäev: 23. märts 2020. a., allikas DZone: https://dzone.com/articles/steps-to-building-authentication-and-authorization

HTTP Methods. (kuupäev puudub). Kasutamise kuupäev: 15. märts 2020. a., allikas RESTfulAPI: https://restfulapi.net/http-methods/

HTTP response status codes. (23. veebruar 2020. a.). Kasutamise kuupäev: 23. märts 2020. a., allikas MDN web docs: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

Interacting with Arduino. (kuupäev puudub). Kasutamise kuupäev: 24. märts 2020. a., allikas Node-RED: https://nodered.org/docs/faq/interacting-with-arduino

Introduction to the server side. (29. november 2019. a.). Kasutamise kuupäev: 31. märts 2020. a., allikas MDN web docs: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction%20

Introduction to web APIs. (31. detsember 2019. a.). Kasutamise kuupäev: 20. märts 2020. a., allikas MDN web docs: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Introduction

Liew, Z. (17. jaanuar 2018. a.). *Understanding And Using REST APIs*. Kasutamise kuupäev: 23. märts 2020. a., allikas Smashing Magazine: https://www.smashingmagazine.com/2018/01/understanding-using-rest-api/

Miessler, D. (23. detsember 2019. a.). *What's the Difference Between a URI and a URL?* Kasutamise kuupäev: 14. märts 2020. a., allikas DanielMiessler: https://danielmiessler.com/study/difference-between-uri-url/

Mishra, H. (17. aprill 2019. a.). *NodeRed Alternatives – Visual programming tools*. Kasutamise kuupäev: 30. märts 2020. a., allikas IoTbyHVM: https://iotbyhvm.ooo/nodered-alternatives/

Nehra, M. (19. september 2019. a.). *Types of APIs | What are APIs? | Different types of APIs*. Kasutamise kuupäev: 17. märts 2020. a., allikas Decipher Zone Softwares: https://dev.to/decipherzonesoft/types-of-apis-what-are-apis-different-types-of-apis-3mjm

Network Client. (kuupäev puudub). Kasutamise kuupäev: 16. märts 2020. a., allikas Network Encyclopedia: https://networkencyclopedia.com/network-client/

Node.js - Introduction. (kuupäev puudub). Kasutamise kuupäev: 24. märts 2020. a., allikas tutorialspoint: https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm

Node-RED forum. (kuupäev puudub). Kasutamise kuupäev: 7. aprill 2020. a., allikas Node-RED: https://discourse.nodered.org/

Node-RED Library. (kuupäev puudub). Kasutamise kuupäev: 7. aprill 2020. a., allikas Node-RED: https://flows.nodered.org/

Node-RED-i kodulehekülg. (kuupäev puudub). Kasutamise kuupäev: 24. märts 2020. a., allikas Node-RED: https://nodered.org/

Nord Pool Spot AS. (kuupäev puudub). *Market Data*. Kasutamise kuupäev: 27. aprill 2020. a., allikas Nord Pool: https://www.nordpoolgroup.com/Market-data1/#/nordic/table

Paessler AG. (kuupäev puudub). *Server*. Kasutamise kuupäev: 20.. märts, 2020. a., allikas Paessler: https://www.paessler.com/it-explained/server

Pedro, B. (7. november 2017. a.). *What are Web APIs*. Kasutamise kuupäev: 16. märts 2020. a., allikas Hackernoon: https://hackernoon.com/what-are-web-apis-c74053fa4072

REST Architectural Constraints. (kuupäev puudub). Kasutamise kuupäev: 14. märts 2020. a., allikas RESTfulAPI: https://restfulapi.net/rest-architectural-constraints/#uniform-interface

Rouse, M. (august 2012. a.). *data point*. Kasutamise kuupäev: 12. märts 2020. a., allikas WhatIs: https://whatis.techtarget.com/definition/data-point

Rouse, M. (aprill 2020. a.). *RESTful API (REST API)*. Kasutamise kuupäev: 8. mai 2020. a., allikas SearchAppArchitecture: https://searchapparchitecture.techtarget.com/definition/RESTful-API

Running Node-RED locally. (kuupäev puudub). Kasutamise kuupäev: 24. märts 2020. a., allikas Node-RED: https://nodered.org/docs/getting-started/local

Running on Android. (kuupäev puudub). Kasutamise kuupäev: 24. märts 2020. a., allikas Node-RED: https://nodered.org/docs/getting-started/android

Sense Tecnic Systems Inc. (kuupäev puudub). *Accounts, Members and Access Control*. Kasutamise kuupäev: 7. aprill 2020. a., allikas FRED - Documentation: http://docs.sensetecnic.com/fred/account-roles/

Sense Tecnic Systems Inc. (kuupäev puudub). *FRED: Front End For Node-RED*. Kasutamise kuupäev: 7. aprill 2020. a., allikas FRED: https://fred.sensetecnic.com/

Siemens AG. (kuupäev puudub). *API Introduction*. Kasutamise kuupäev: 31. märts 2020. a., allikas Climatix IC Cloud API: https://portal.api.climatixic.com/Introduction

Siemens. (kuupäev puudub). *Climatix IC - Cloud solution for OEMs*. Kasutamise kuupäev: 20. aprill 2020. a., allikas Siemens: https://new.siemens.com/global/en/products/buildings/hvac/oem/remote-control-climatixic.html

Solcast. (2019). *Global solar irradiance data and PV system power output data*. Kasutamise kuupäev: 27. aprill 2020. a., allikas Solcast: https://solcast.com/

Total.js koduleheküg. (kuupäev puudub). *Total.js Flow*. Kasutamise kuupäev: 30. märts 2020. a., allikas Total.js: https://www.totaljs.com/flow/

Using HTTP Methods for RESTful Services. (kuupäev puudub). Kasutamise kuupäev: 14. märts 2020. a., allikas RestApiTutorial: https://www.restapitutorial.com/lessons/httpmethods.html

What is REST API design? (kuupäev puudub). Kasutamise kuupäev: 8. märts 2020. a., allikas Mulesoft: https://www.mulesoft.com/resources/api/what-is-rest-api-design

What is REST? (kuupäev puudub). Kasutamise kuupäev: 8. märts 2020. a., allikas RESTfulAPI: https://restfulapi.net/