TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Mari-Liis Kaldvee

Cumulocity platvorm

Bakalaureusetöö (9 EAP)

Juhendajad: Alo Peets, Anne Villems, Taavi Duvin

Tartu 2017

Cumulocity platvorm

Lühikokkuvõte:

Käesoleva bakalaureusetöö eesmärgiks on anda ülevaade IoT-platvormi Cumulocity kohta ning kasutada Cumulocity keskkonda ja võimalusi lihtsa IoT lahenduse näitel, milleks on sensoritega äratuskella prototüüp, mis suhtleb Cumuocity API-ga. Äratuskella ekraanil kuvatakse temperatuur, mis pärineb temperatuuri sensorilt ning kohalik aeg. Cumulocity keskkonda saadetakse sensorite tulemused ning pikemaajaliseks andmete säilitamiseks saadetakse Zapieri vahendusel need Google Spreadsheeti. Samuti luuakse esipaneel Cumulocity keskkonda relevantse infoga mõõtmistulemustest ning komponent, mille kaudu võimaldatakse luua uusi alarme Arduino seadmele läbi Cumulocity.

Võtmesõnad:
IoT, AngularJS, Arduino, Cumulocity, Node.js, GSM-moodul

CERS:
P170 (Computer science, numerical analysis, systems, control)

Cumulocity platform

Abstract:

The aim of this bachelor’s thesis is to give an overview of the IoT platform Cumulocity and use the Cumulocity environment in creation of an alarm clock with sensors that communicates with the Cumulocity API. On the screen of the alarmclock, there is local time and temperature that originates from the temperature sensor. The sensor readings are sent to Cumulocity and in order to preserve the data longer, the data is sent to Google Spreadsheet with Zapier. Also, there is dashboard in the Cumulocity environment with relevant information on the measurement data and a component that enables the creation of alarms to the Arduino device.

Keywords:
IoT, AngularJS, Arduino, Cumulocity, Node.js, GSM-module

CERCS:
P170 (Computer science, numerical analysis, systems, control)

Sisukord

Sissejuhatus

Internet of Things (IoT) ehk asjade internet (samuti eestipärase nimetusega värkvõrk) on viimase aja üks trendikamaid tehnoloogia teemasid, mis on paljude ettevõtete tähelepanu osaliseks saanud ning millele nähakse suurt kasvu tulevikus. Asjade internetiks nimetatakse igapäevaseadmete ühendamist internetiga, et vahetada informatsiooni teiste seadmete või kasutajatega ning luua selle kaudu mingit lisandväärtust. Näiteks targad termostaadid nagu Nest [1] kasutavad seadme sensoreid, reaalaja ilmaennustusi, elektri börsihinda ja teadmist elanike asukohast, et tegutseda optimaalsemalt ning säästa elektrit ja samuti võimaldab kasutajal kaugjuhtida seadet/seadmeid vastavalt vajadusele.

IoT tulekut ennustati mitukümmend aastat tagasi, kuid põhjus miks IoT nüüd on aina rohkem ettevõtete ja meedia tähelepanu saanud, seisneb selles, et suuremahuliste IoT süsteemide loomine on muutunud odavamaks kui varem. Seoses IP-protokolli Ipv6 kasutuselevõtuga on suurenenud unikaalselt identifitseeritavate aadressite hulk, mis suudaks toetada ka hinnanguliselt aastaks 2020 võrku ühendatud üle 30 miljardi IoT seadme [2]. Samuti on positiivne olnud pilvetehnoloogia, mis võimaldab vähendada riistvara kulusid andmete salvestamiseks ning andmetöötluseks vajalike arvutuste läbiviimiseks [3].

IoT-s nähakse võimalust vähendada ressursside raiskamist, jälgida ja koguda relevantseid andmeid, optimeerida seadmete hooldustööde läbiviimist tehastes. Kõige suuremat kasu nähakse IoT lahenduste kasutuselevõttu tootmissektoris (läbi operatsioonide halduse ning ennetava paranduse), linnades (liikuse reguleerimine, ressursside haldus, kodanike tervis ja turvalisus), jaekaubanduses (iseteeninduspunktid) ning liiklussüsteemides (logistika ruutimine, autonoomsed sõidukid, navigatsioon) [4].

Cumulocity on IoT pilveplatvorm, mis võimaldab jälgida oma ühendatud seadmete andmeid ning kasutaja saab ära kasutada erinevaid API-sid ja Cumulocity rakendusi, mille kaudu saab hallata oma seadmeid ja IoT lahendusi, visualiseerida andmeid, kaugjuhtida oma seadmeid ning luua ja hallata oma rakendusi. Selliste platvormide kasu seisneb selles, et kasutaja mured IT-infrastruktuuri suhtes on suunatud teenusepakkujale, kes ise hoolitseb turvalisuse, ühendatuse ja andmete säilivuse peale [5].

Bakalaureusetöö eesmärgiks on kirjeldada ning rakendada Cumulocity funktsionaalsusi anduritega äratuskella prototüübi näitel. Täpsemalt luuakse lahendus, et seade saadab platformile niiskuse- ja temperatuurimõõtmised ning platform saadab seadmele operatsiooni kui alarm peaks tööle minema või kui seade peaks saatma mõõtmistulemused keskkonda.

Mõõtmistulemused saadetakse vahendaja kaudu pikaajalisemaks salvestamiseks Google Spreadsheeti ning alarmiaegade valimine käib läbi kasutaja loodud komponendi. Selle lahenduse näitel tutvustatakse spetsiifilisemalt, mida Cumulocity pakub kasutajatele oma IoT-lahenduste loomiseks. Kuna IoT on suhteliselt abstraktne mõiste, siis sai otsustatud võtta seade, mida paljud inimesed igapäevaselt kasutavad ning selle põhjal teha üks IoT-lahendus, mis

kasutaks ära võimalikult palju erinevaid tahkusid sellest, mida Cumulocity pakub oma kasutajatele.

Käesoleva töö esimene peatükk tutvustatakse Cumulocity platvormi ning põhilisi funktsionaalsuseid. Teises peatükis kirjeldatakse seadme riistvara ning Arduino programmi erinevaid osasid ja vajalikke teeke. Kolmas peatükk kirjeldab esipaneelis kasutatud Cumulocity komponente ning ise komponendi loomist äratuskellale alarmi lisamise komponendi näitel. Neljas peatükk näitab, kuidas saata andmeid Zapieri vahendusel Google Speadsheeti.

Cumulocity tutvustus

Üldtutvustus

Cumulocity on IoT platvorm, millega saavad kasutajad oma seadmeid ühendada, teostada järelvalvet ning lisada loogikat seadmetele või kasutada seadmetest tulevat infot välissüsteemides. Nt. oma nutitelefoni ühendamisel platvormiga, on võimalik saada Cumulocity keskkonnas infot seadme signaalitugevuse, sensorite ning asukoha kohta. Samuti on võimalik luua niiöelda "tarku reegleid", mis käivituvad mingil sündmusel (nt. kui ühendus platvormi ja seadme vahel katkes, siis saadetakse e-mailile sõnum selle kohta vms). Platvormi nimi on kokku pandud kahest sõnast "Cumulus" (tähendusega "pilv") ja "Velocity" (tähendusega "kiirus"), mis sümboliseerib ettevõtte eesmärki klientidele pakkuda kiireid pilvelahendusi [6].

Cumulocityga saab infot seadmete ja ühenduse kohta ning hallata seadmete konfiguratsiooni, tarkvara ja püsivara. Samuti pakub Cumulocity graafikuid seadme erinevate sensorite mõõtmistulemustest ning võimaldab kasutajal mugandada oma keskkonda vastavalt oma vajadusele (nt. luua alarmireegleid, lisada reaalajas töötavat äriloogikat või luua graafikuid andmetest) [5].

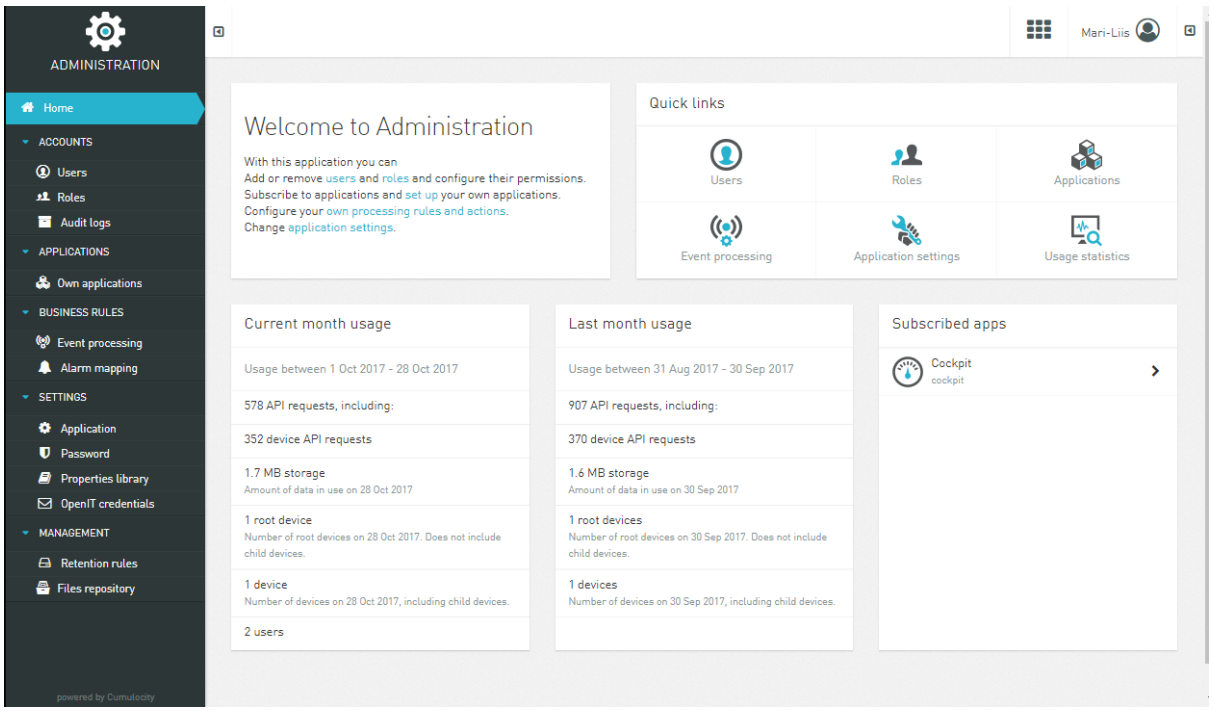
Cumulocity toetab kõiki suurimaid arenduskomplekte nagu Arduino, Raspberry Pi, Tinkerforge jne. Ettevõttel on ka geneerilisi klienditeeke Java, JavaME, C/C++ ja Lua jaoks ning samuti saab platvormi ja seadme vahel suhtlemiseks kasutada API-liidest [5]. Peale selle oma Javascripti teegid, mis võimaldab luua Cumulocity platvormiga sarnaseid kujundusi ning suhelda API-ga.

Põhirakendused

Igal kasutajal on kolm põhirakendust oma keskkonnas- seadmehalduse, kokpiti ja administratsiooni rakendus.

Administratsiooni rakenduse (vt joonis 1) funktsionaalsused:

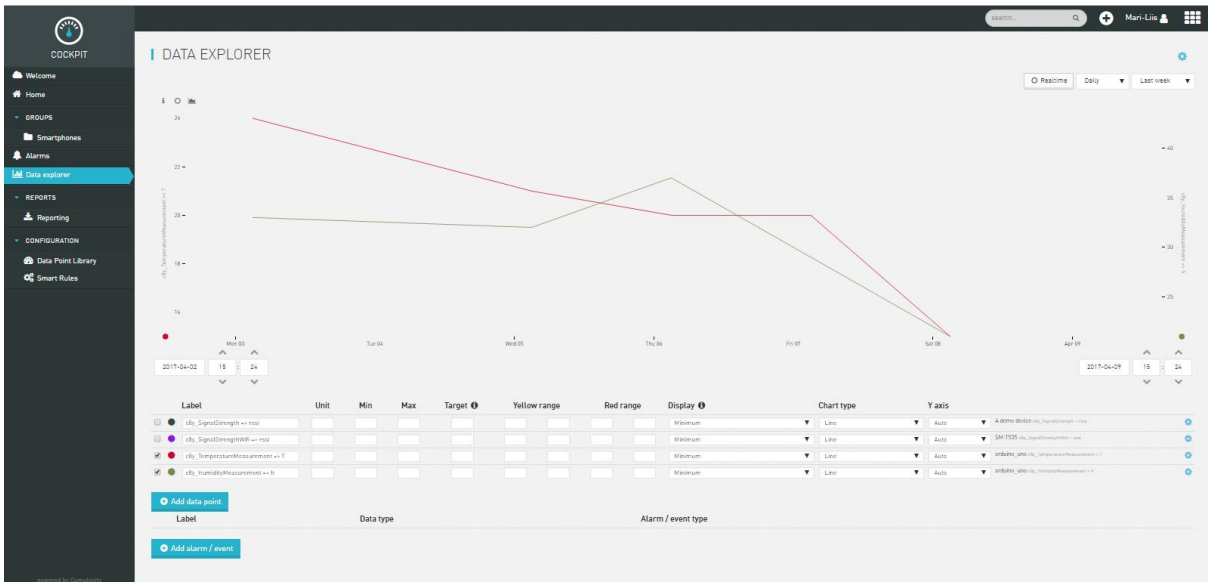
- Saab lisada ja eemaldada kasutajaid ning rolle
- Saab uusi rakendusi oma keskkonda lisada
- Saab rakenduste sätteid muuta
- Saab vaadata sündmuste logi
- Saab hallata olemasolevaid CEP-mooduleid (inglise keeles Cumulocity Event Processing) reaalajaloogika lisamiseks
- Saab muuta alarmitüüpide tõsisust
- KPI-de (inglise keeles Key Performance Indicator) lisamine ja vaatamine
- Saab hallata erinevate andmete säilivuse perioodi
- Saab lisada faile, neid alla laadida ja kustutada



Joonis 1: Administratsiooni rakenduse vaade

Kokpiti rakenduse (vt joonis 2) funktsionaalsused:

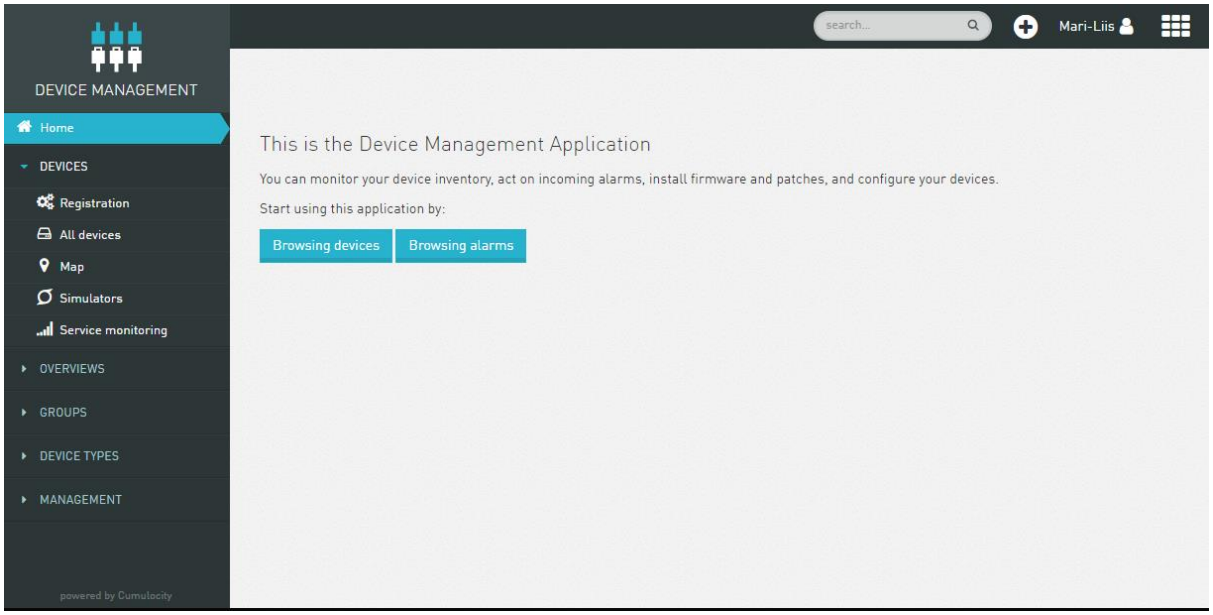
- Saab hallata “tarku” reegleid
- Saab lisada uusi seadmete gruppe
- Saab seadmete mõõtmistulemusi visualiseerida
- Saab vaadata hetkel aktiivseid alarme ja selle detaile
- Saab aktiivsete alarmide staatust muuta
- Saab esipaneele luua ja personaliseerida komponentide lisamise ja seadistamisega
- Saab luua raporteid mingile seadmele või/ja ajaperioodile



Joonis 2: Kokpiti rakenduse vaade

Seadmehalduse rakenduse (vt joonis 3) funktsionaalsused:

- Saab seadmeid registreerida ja kustutada
- Saab luua simulaatoreid, mis jäljendavad tavalisi seadmeid
- Saab seadmegruppe luua
- Saab hallata seadme konfiguratsiooni
- Saab seadme kohta ülevaate
    - Seadme tehniline info
    - Seadmelt saadud mõõtmistulemuste graafikud
    - Seadme aktiivsed alarmid
    - Ülevaade viimastest operatsioonidest ja sündmustest
    - Seadme geograafiline asukoht



Joonis 3: Seadmehalduse rakenduse vaade

API-liides ja "targad" reeglid

API (inglise keeles Application Programming Interface) ehk rakendusliides on programm, mis sisaldab endas reeglistikku, millega üks programm saab teise programmi (nt. mingi veebi- või mobiilirakenduse) teenustele või infole päringuid teha. API-dele on iseloomulik, et päringute ja nende vastuste esitus on kindlaid süntaksireegleid järgides. Näiteks on väga levinud JSON-i (inglise keeles JavaScript Object Notation) kasutamine selleks.

JSON-I kujul on minimalistliku tekstiformaadiga, mida on inimestel kerge lugeda ja kirjutada ning masinatel lihtne genereerida ja sisse lugeda [7]. Ka Cumulocity kasutab JSON-it oma rakendusliideses ning oma andmetüüpide kirjeldamisel.

Näide Cumulocity sündmuse andmetest JSON-i formaadis:

```
{
      "source": {
      "id":"10200" },
     "type": "TestEvent",
     "text": "sensor was triggered",
     "time": "2014-03-03T12:03:27.845Z"
}
```

Antud näide on sündmuse andmetüüp JSON-I kujul, mis sisaldab infot selle kohta, mis seade sündmuse tekitas ning mis tüüpi sündmusega tegemist on (antud juhul on sündmuse tüübi väärtuseks "TestEvent") ning samuti salvestatakse JSON-i struktuuri ka aeg, millal sündmus tekkis. JSON-ile on omane paaride kasutamine, kus paari esimene liige on sõne ning paari teine liige on selle tunnuse väärtus, mis võib olla näiteks number, järjend, objekt, tõeväärtus või sõne. Nt. selles näites oli tunnuse "source" väärtus objekt, mis sisaldas endas tunnust "id", mille väärtus oli "10200".

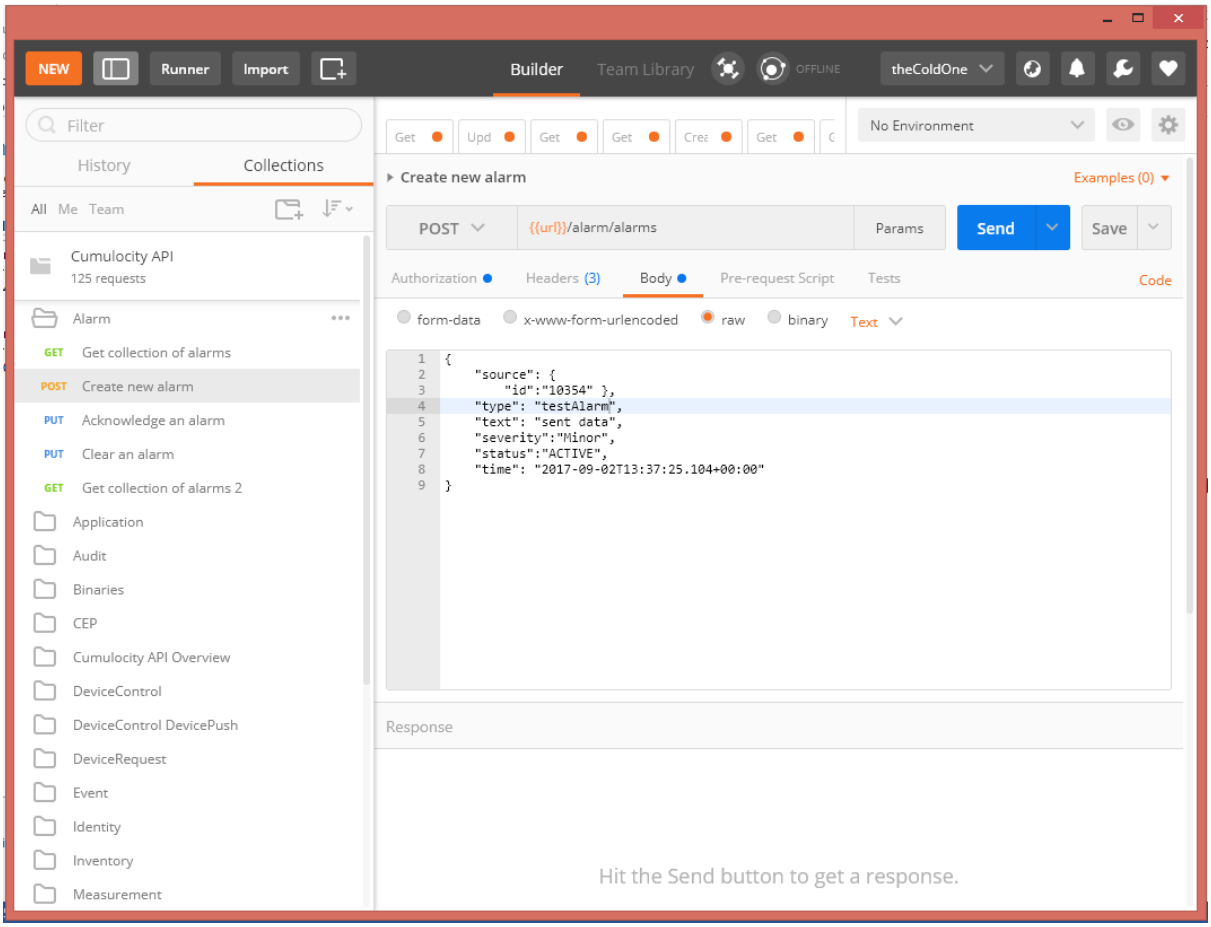
Cumulocity kasutab REST (inglise keeles representational state transfer) rakendusliidest, mis on üks populaarseimaid API-tüüpe. Seda iseloomustab HTTP (inglise keeles Hyper Text Transfer Protocol) ja HTTPS (mis on protokoll, mis kasutab krüpteeritud HTTP ühendust) klient-server protokollide kasutamine ning nende olekuvabasus (inglise keeles statelessness), mis tähendab, et igat päringut käsitletakse sõltumatutena teistest päringutest [8]. See võimaldab saavutada kõrgemat skaleeritavust ning kiiremat suhtlust kliendi ja server vahel.

REST-i rakendusliidesed kasutavad HTTP protokolli kõige rohkem järgmisi meetodeid [9]:

1. GET – teeb pöördumise andmete kättesaamiseks
2. DELETE – teeb pöördumise ressursi kustutamiseks
3. PUT – teeb pöördumise mingi ressursi muutmiseks
4. POST – teeb pöördumise uute andmete vastu võtmiseks ja salvestamiseks süsteemi

Cumulocity'l on API-liides, millega võimaldab suhtlust Cumulocity platvormi ja seadme vahel. Nt. saab selle kaudu hallata seadmeid ja nendega seonduvat infot. Samuti saab saata uusi mõõtmistulemusi, aktiveerida või maha võtta alarme, alustada, lõpetada operatsioone või lisada reaalajas rakendatavat loogikat jne. Cumulocity soovitab API-liidesega tutvuda Postmani kaudu, mis on rakendus, mis teeb API-liidestega suhtlemise ning päringute genereerimise lihtsamaks. Selleks, et pöördumisi Cumulocity API-liidesele teha, oleks vaja autentimiseks oma Cumulocity kasutajanime ning parooli.

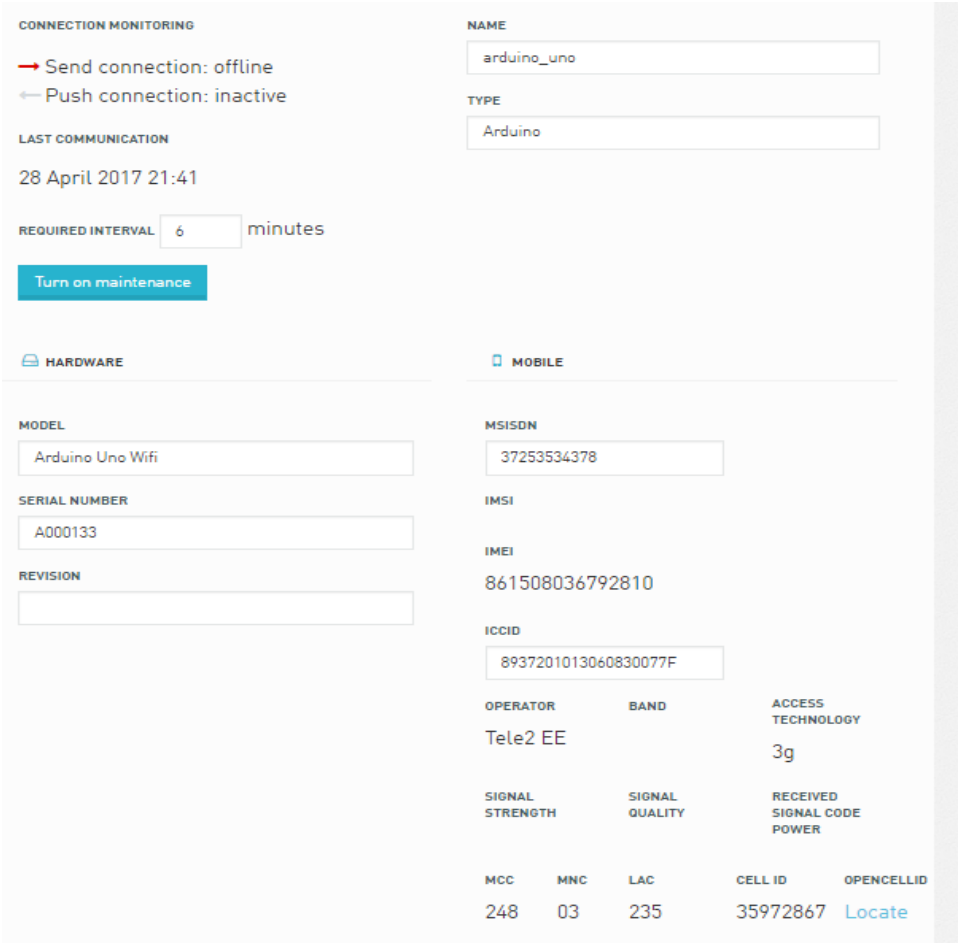
Postman teeb Cumulocity API kergemini arusaadavamaks ning võimaldab algajal Cumulocity kasutajal parema ettekujutuse saada erinevatest võimalustest, milliseid pöördumisi on võimalik Cumulocity keskkonnale teha. Näiteks joonisel 4 on näha alarmide võimalike pöördumisi nagu uue alarmi loomine (joonisel suuremas redaktoris on näha JSON-i objekti, mis sisaldab uue alarmi infot), kustutamine ning olemasolevate alarmide kättesaamine.



Joonis 4: Postmani vaade

API-liideses on esindatud 6 erinevat pöördumiste rühma - inventuuri, sündmuste, mõõtmiste, seadme kontrolli ning alarmide pöördumised.

Inventuuris on seadmete informatsioon nagu mis on iga seadme unikaalne identifikaator Cumulocity süsteemis, nimi, tüüp, omaniku andmed ning nt. millal seadet viimati uuendati vms. Samuti saab lisada nt. seadme koordinaate, milliseid mõõtmisi seade peaks lubama, riistvara mudeli infot ning mis võrgus seade töötab jne. API-liides annab ka infot selle kohta, kas seade on hetkel Cumulocityga ühendunud ning millal viimati ühendus saavutati. Suur osa sellest infost on vaadeldav seadmehalduse vaates (joonis 5).



Joonis 5: Seadme info

Sündmused sisaldavad järgmist infot:

- Sündmuse id
- Aeg, mil sündmus jõudis platvormi
- Sündmuse tüüp
- Tekst (üldiselt selleks, et anda täpsem kirjeldus sündmusest)
- Aeg, mil sündmus oli loodud
- Seadme id Cumulocitys, mis sündmuse esile kutsus

Näiteks luuakse platvormis sündmus kui mingile seadmele on määratud uus asukoht vms.

Mõõtmistulemused sisaldavad järgmist infot:

- Mõõtmise id
- Mõõtmise tüüp
- Aeg, mil mõõtmine sooritati seadme poolt
- Seadme id Cumulocitys, mis mõõtmise saatis
- Mõõtmise ühik
- Mõõtmise väärtus

Näiteks kui tahta sensori mõõtmistulemust saata platvormi, siis peaks kirjeldama, mis tüüpi mõõtmistulemus on, mis mõõtmistulemuse ühik on ning mis väärtus sellel on. Kasutades Postmani peaks kasutaja ise ülejäänud info ise täitma, kuid kasutades nt. Cumulocity teeki Arduinole, siis ülejäänud info täidetakse kasutajale ise ära.

Operatsioon sisaldab järgmist infot:

- Operatsiooni id
- Aeg, mil operatsioon loodi
- Operatsiooni staatus
- Seadme id, millele saadetakse operatsioon täitmiseks

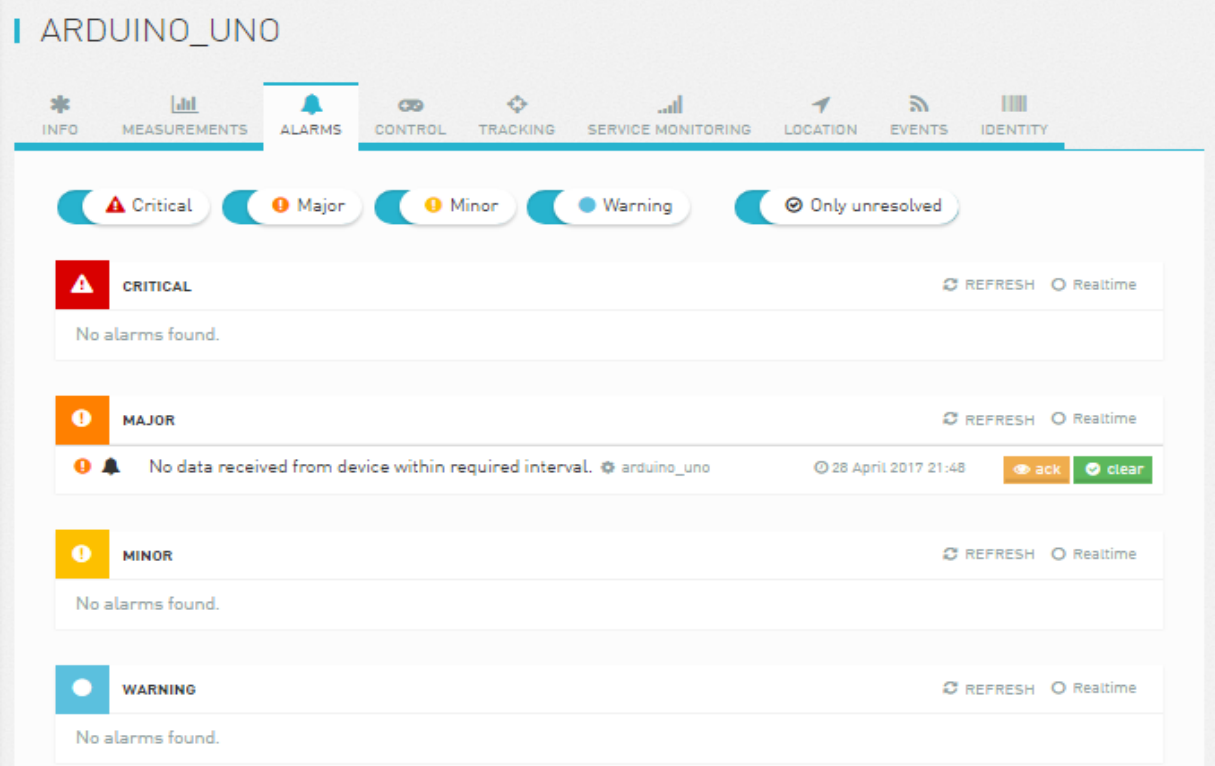
Seadme kontrollimiseks ning tegevuste saatmiseks luuakse operatsioone, millel on seadme info, mida peaks operatsiooni läbi viima ning samuti operatsiooni staatus (nt. kas tegemist on lõpetatud või veel pooleli oleva operatsiooniga). Nt. saab lasta seadet taaskäivitada või LED-tulesid vilkuma panna operatsioonide kaudu.

Üks operatsiooni alaliik on ka hulkoperatsioon (ingl. keeles bulk operation), millega saab saata sama operatsiooni mitmele seadmele oma valitud kellaajaga. See võimaldab planeerida erinevaid operatsioone ning mugavamalt hallata seadmegruppe, mis peaksid käituma sarnaselt. Hulkoperatsiooni juures peab samuti määrama ära kui pikk on ajavahe pärast iga operatsiooni loomist.

Alarm sisaldab järgmist infot:

- Alarmi id
- Aeg, mil alarm oli jõudnud platvormi
- Alarmi tüüp
- Alarmi tõsisus
- Alarmi staatus
- Alarmi kirjeldav tekst
- Aeg, mil alarm oli loodud
- Seadme id, mis on alarmiga seotud

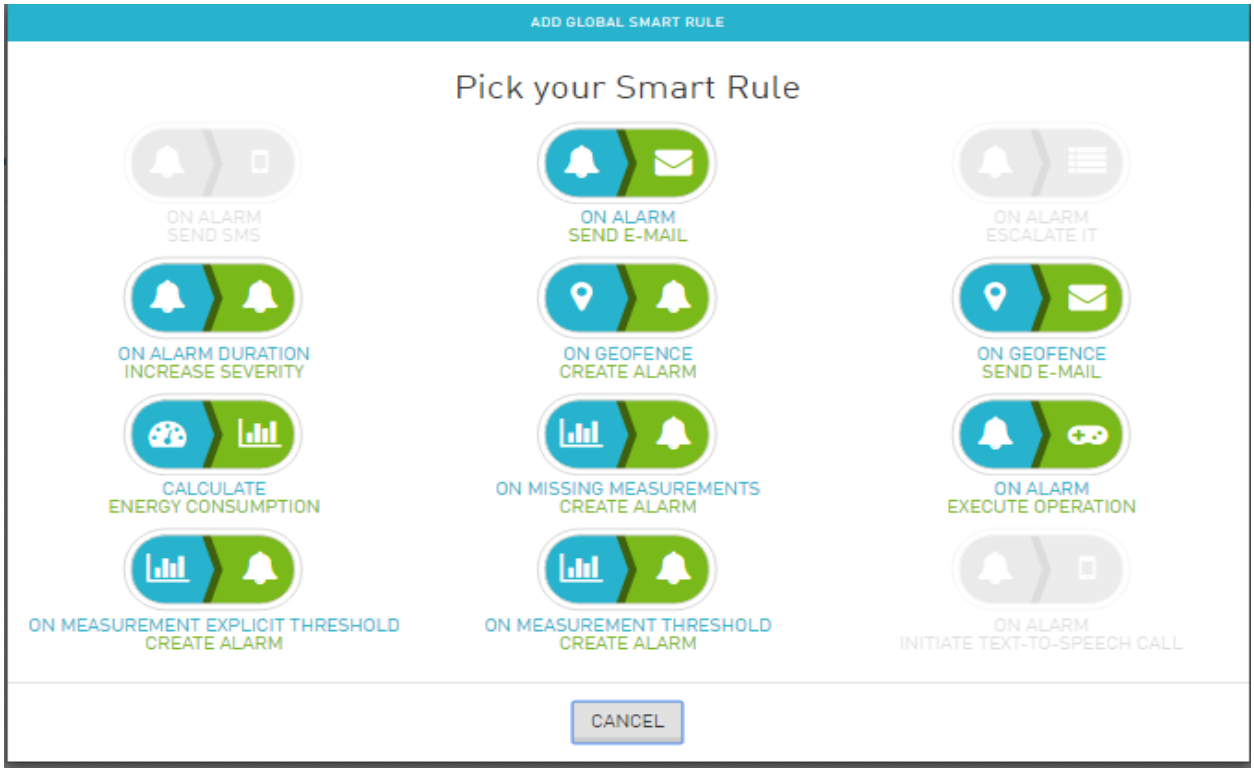
Alarm võib nt. käivituda kui seade pole kättesaadav või kui seade on lahkunud mingilt geograafiliselt alat või kui mingi anduri mõõtmistulemus on jõudnud defineeritud läveni või üldse puudub. Kasutaja saab ka luua reegleid selleks, et platvorm ise suudaks reageerida erinevatele alarmidele. Aktiivseid alarme on võimalik ka näha seadmehalduses, kus need on kategoriseeritud vastavalt alarmi kriitilisusele (joonis 6).



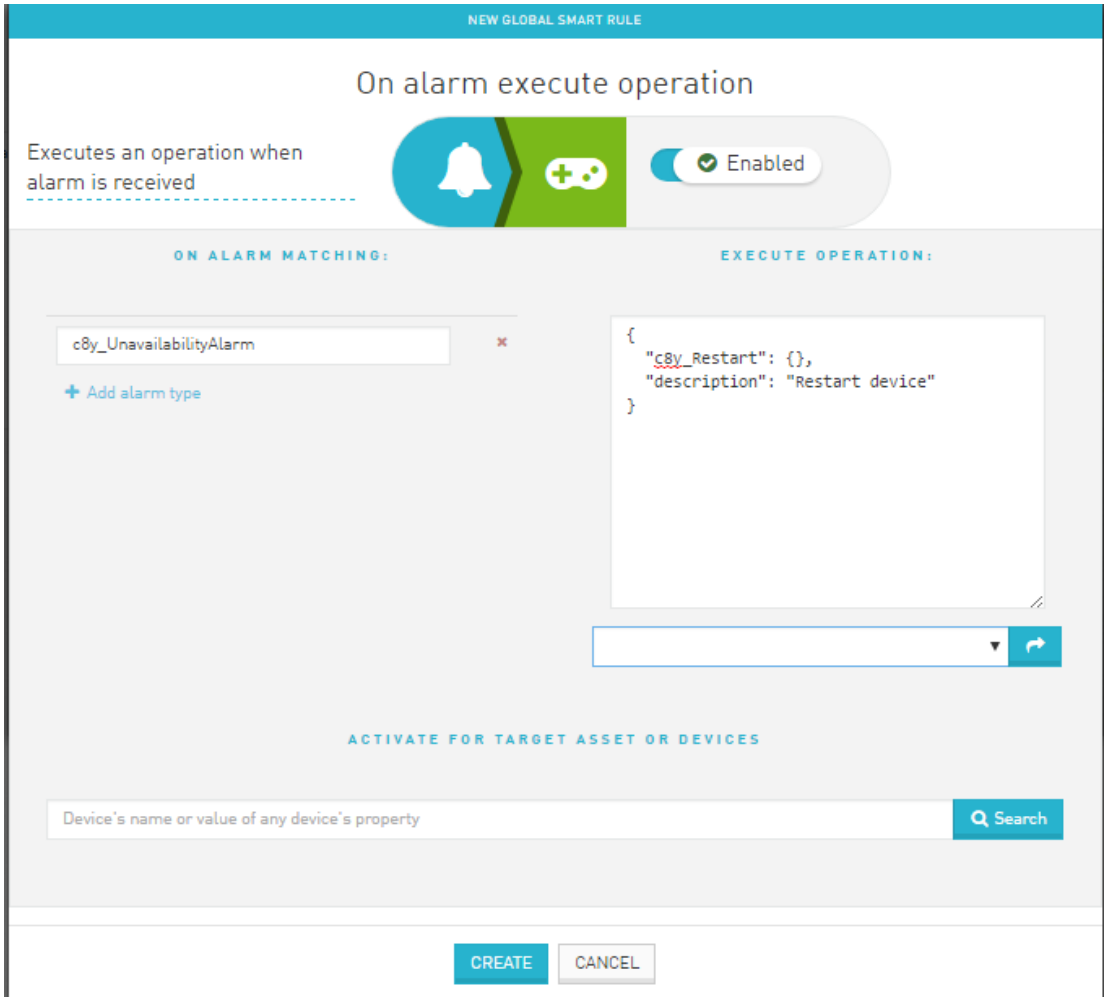
Joonis 6: Alarmide vaade

Nt. võib kasutaja luua reegleid, mis alarmi käivitumisel saadaksid SMS-sõnumi või e-kirja kasutajale, teavitades teda olukorrast ning võimaldada ise otsustada, mida ta edasi teha sooviks või siis alarmi tekkides automaatselt luua reegel, mis ise tegeleb situatsiooniga (nagu konkreetse alarmi käivitumisel luua mingi operatsioon pärast mille läbiviimist kustutataks alarm ära).

Platvorm võimaldab kergelt ise reaalajas töötavaid “tarku” reegleid (joonis 7) luua kokpiti rakenduses, kus neid saab ka muuta ja kustutada. Reegleid on võimalik seadistada nii üksikutele seadmetele kui ka seadmete grupile. Nt. spetsiifilise alarmi korral mingi operatsiooni saab käivitada kui lisada “targa” reegli liideses alarmi tüüp (nt et tegemist on “c8y_UnavailabilityAlarmiga”) ning operatsiooni juurde lisada JSON-i objekt operatsioonist või otsida sobiv operatsioon valikväljast (joonis 8, kus on valitud seadme taaskäivitamise operatsioon).



Joonis 7: Tarkade reeglite vaade

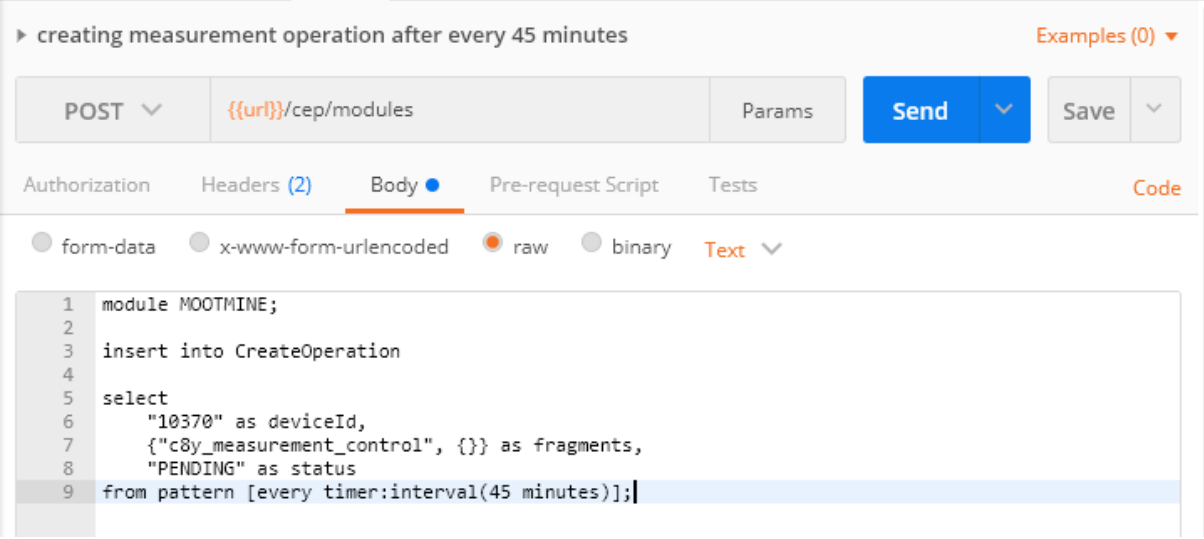


Joonis 8: Targa reegli loomine

Reaalaja moodul

Parema paindlikkuse saavutamiseks on peale reaalajas töötavate "tarkade" reeglite ka võimalik ise keerulisemaid tegevusahelaid luua ning andmeid töödelda vastavalt vajadusele. Platvorm on loonud SQL-i laadse keele CEL (Cumulocity Event Language), mida saab kasutada ka API-liidese kaudu. Seal ja ka administratsiooni rakenduse kaudu saab hallata CEP-mooduleid, mis tegelevad kasutajate poolt defineeritud reaalajas töötavate toimingutega ning sisaldab ka "tarku reegleid" [10].

CEL-keele näitlikustamiseks toob autor näite, kuidas luua CEP-moodulit, mis loob iga 45-minuti tagant operatsiooni seadmele (joonis 9). Esimesel real on mooduli nimi, mis näites on "MOOTMINE". Järgnevalt lisatakse voogu "CreateOperation" uus operatsioon iga 45 minuti tagant ning lisatakse seadme identifikaator (antud näitel on see "10370" ehk operatsioon saadetakse seadmele selle id-ga), JSON-i objekti fragmendina ning operatsiooni staatuse pannakse "PENDING", et anda indikatsioon sellest, et tegemist on veel täitmist ootava operatsiooniga. Joonise 9 alumisest tekstialast on näha, et pöördumine platvormile on tehtud ja CEP-moodul on "DEPLOYED" ehk aktiivne.



Joonis 9: Postmani vaade. CEP-mooduli loomine

Näiteks saab iga kindla ajaperioodi järel aktiveerida mingit operatsiooni, mis saadetakse seadmele täitmiseks ning luua alarme kui antud operatsioon ei ole mingi aja jooksul täidetud. Samuti on võimalik kuulata erinevaid voogusid (nt. loodud alarmide voog või uuendatud operatsioonide voog) ning lisada reegleid, mis reageerivad kindlat tüüpi alarmidele kui need on aktiveerunud vms.

Seadmetest tulenev informatsioon läbib töötluse ning kui tegemist on püsiva infoga, siis salvestatakse see andmebaasi. Samuti on võimalik, et informatsiooni põhjal käivituvad mingid reeglid, mille korral saadetakse mingi korraldus seadmele või teave situatsiooni kohta nt. e-mailile või hoopis saadetakse välissüsteemi Zapieri kaudu.

Platvormivälised tegevused

Oma keskkonda on võimalik ühendada Zapieriga, et luua erinevaid integratsioone teenuste vahel. Zapier on ettevõte, mis pakub teenust, millega automatiseerida suhtlust erinevate rakenduste vahel. See toetab üle 500 veebiteenuse, kaasaarvatud Google'i teenuseid nagu Google Sheets, Gmail ning teisi populaarseid teenuseid nagu Facebook Pages ja Slack [11].

Nt. Cumulocity kasutaja saab alarme saata ettevõtte CRM süsteemi või edastada mõõtetulemusi arvutustabelisse vms. Infot on võimalik saata mõlemapoolselt - Cumulocity keskkonnast Zapieri keskkonda ja Zapieri keskkonnast Cumulocity keskkonda. Teenuse kasutamiseks on vajalik lisaks Cumulocity kasutajale ka Zapieri kasutajatunnus [12].

Kasutaja rakendused ning komponendid

Kasutajad saavad ka enda Cumulocity keskkonnale teha oma kasutajaliidese komponente ning rakendusi ja neid oma keskkonda üles panna. Cumulocity platvormil on erinevaid komponente, mida saab lisada ja isikupärastada vastavalt kasutaja maitsele nt. kokpiti rakenduse esipaneelis (inglise keeles dashboard). Kasutajal on ka võimalus rakenduste ja komponentide

arendamiseks kasutada Javascripti raamistikke Node.js-i ja Angular.js ning Cumulocity SDK-d (inglise keeles Software Development Kit) [13].

Platvormil on oma SDK Javascriptile, kus on teegid C8y.Core ja C8y.UI. Esimene täidab sarnaseid funktsionaalsuseid Cumulocity API-le ning C8y.UI on selleks, et panna paika nt. komponendi paiknemine kasutajaliideses (nt. kas komponent on kättesaadav vidinate menüüst või sätete alla on tehtud eraldi leht komponendile)[13].

Oma komponentide ja rakenduste loomiseks on vaja kasutada Node,js-i, mis on üks populaarsemaid Javascripti raamistikke. Tegemist on vabavaralise Javascripti raamistikuga, mis võimaldab Javascripti koodi jooksutada ka serveri poolel. Node.js-i paketihaldussüsteemiks on vaikimisi npm, mis võimaldab arendajatel kergemini avaldada ja jagada erinevate teekide lähtekoodi ning lihtsustada teekide allalaadimist, uuendamist ja eemaldamist.

Cumulocity-l on oma Node,js-i teek cumulocity-tools, mille peaks enne arendustööd alla laadima npm-iga. Antud teek aitab lihtsustada komponentide ja rakenduste ülessättimise (nt. genereerides automaatselt algse failistruktuuri) ning loob sihtkausta, mida oma keskkonda lisada ja samuti saab automaatselt juurutada rakenduse keskkonda. Samuti saab selle kaudu Cumulocity teekide viimased versioonid alla laadida oma projektile [14].

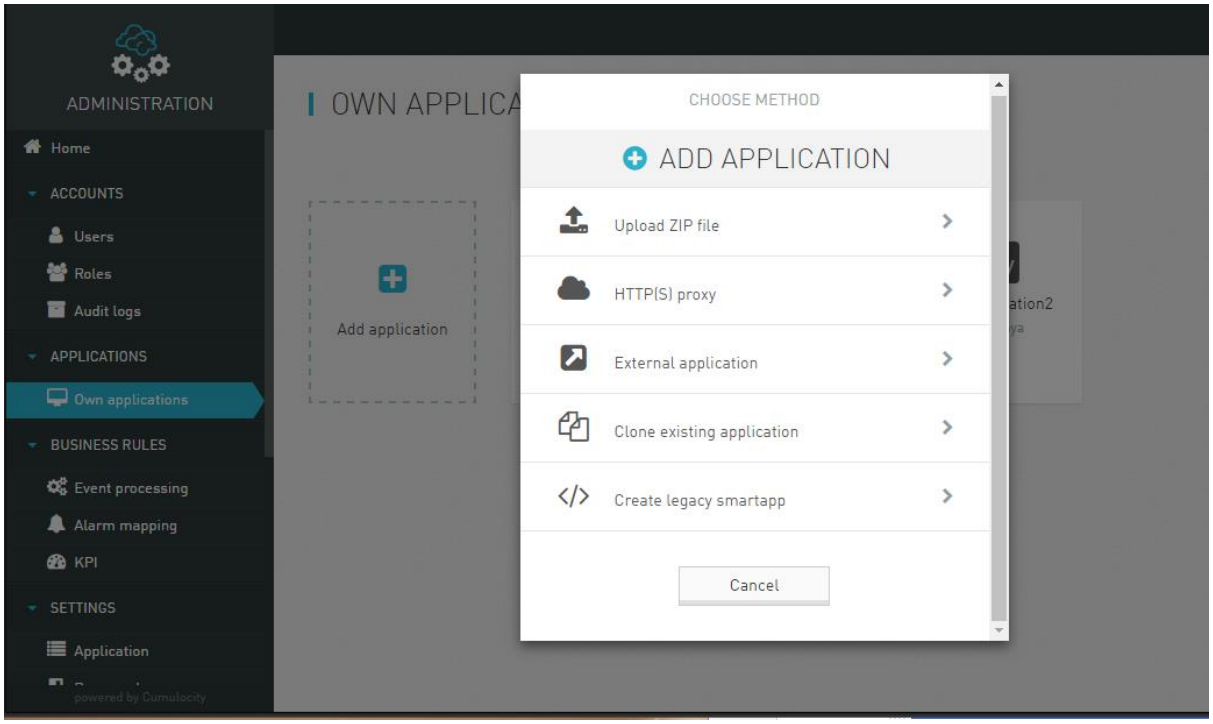
Angularjs on vabavaraline Javascripti raamistik, mida peamiselt arendab Google. Seda kasutatakse peamiselt üheleherakenduste (inglise keeles Single Page Application) loomiseks. Angularjs-i peamised printsiibid on modulaarsus erinevate koodiosade lahushoidmiseks, testitavus ning koodibaasi kerge hallatavus. Üks positiivne omadus on kahepoolne andmesidumine vaate ja mudeli vahel, mis tagab, et kui ühes pooles midagi muutub, siis ka teisel pool toimub muutus. See toimub skoobi vahendusel, mis on nii vaate kui kontrolleri jaoks kättesaadav [15].

Rakendused ning kompondendid peavad olema kindla failistruktuuriga (joonis 10) ning sisaldama cumulocity.json faili, mis sisaldab rakenduse nime, viiteid komponentidele ning rakenduse võtit jne. Ka komponentidel on oma cumulocity.json fail, mis sisaldab viiteid erinevatele kasutatud failidele, komponendi nime ja kirjeldust jne.

```
<<root folder>>
├── cumulocity.json
│    ...
└── plugins
        └── <<plugin name>>
                ├── cumulocity.json
                └── index.js
            ...
```

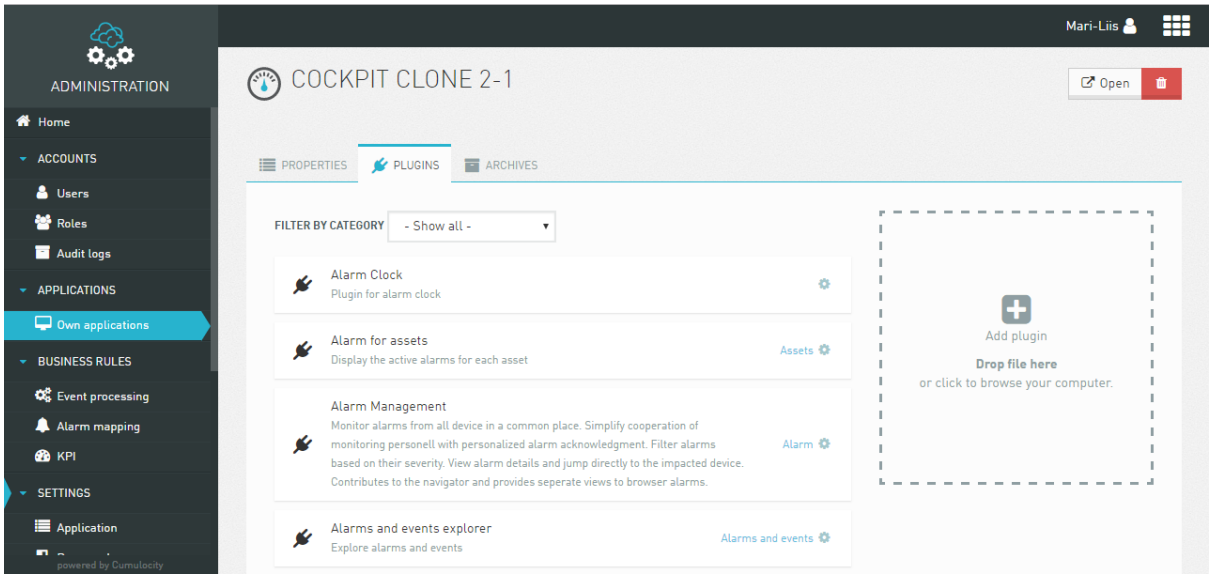
Joonis 10: Cumulocity rakenduse projekti struktuur

Cumulocity keskkonda rakenduste ülesse panemiseks on erinevaid võimalusi. Esiteks võib seda juurutada Cumulocity Node.js teegiga. Teiseks, on võimalus minna administratsiooni rakendusse (joonis 11) ning sealt kaudu laadida ZIP-fail rakendusega, mida saab cumulocity-tools teegiga genereerida. Administratsiooni rakenduses on võmalik ka suunata mingile leheküljele, kus rakenduse kood peaks olema. Samuti on võimalik teha koopia olemasolevast rakendusest ning lisada tollele uusi komponente vms.



Joonis 11: Rakenduse lisamine Cumulocity keskkonda

Komponentide lisamiseks mingile rakendusele peaks administratsiooni rakenduses (joonis 12) võtma lahti konkreetse rakenduse sätted ning seal saab hallata nii komponentide lisamist kui eemaldamist. Cumulocity npm teegiga saab genereerida komponendile eraldi ZIP-faili, mille saab lisada rakenduse sätete vaates.



Joonis 12: Rakenduse komponentide vaade

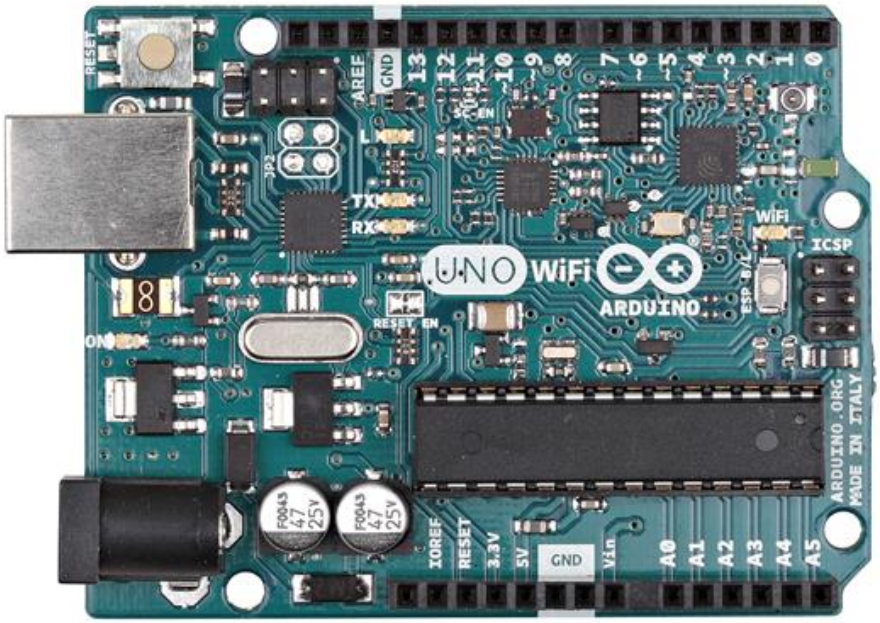
Sensoritega äratuskell

Käesolevas peatükis kirjeldadakse kasutatud riistavara ning Arduino programmis kasutatud teeke ning nende kasutust programmis. Nii Arduino kood kui ka Cumulocity CEP-moodulide koodid ja komponendi kood asuvad Githubi repositooriumis [16].

Arduino Uno Wifi

Antud alapeatükk põhineb materjalidel, mis pärinevad Arduino kodulehelt [17] [18] ja Wikipediast [19].

Arduino Uno on üks kõige populaarsemaid Arduino arendusplaate ning Arduino Uno Wifi mudelil (joonis 13) on ka Wifi moodul, mis võimaldab seadme Wifi-võrku ühendada.



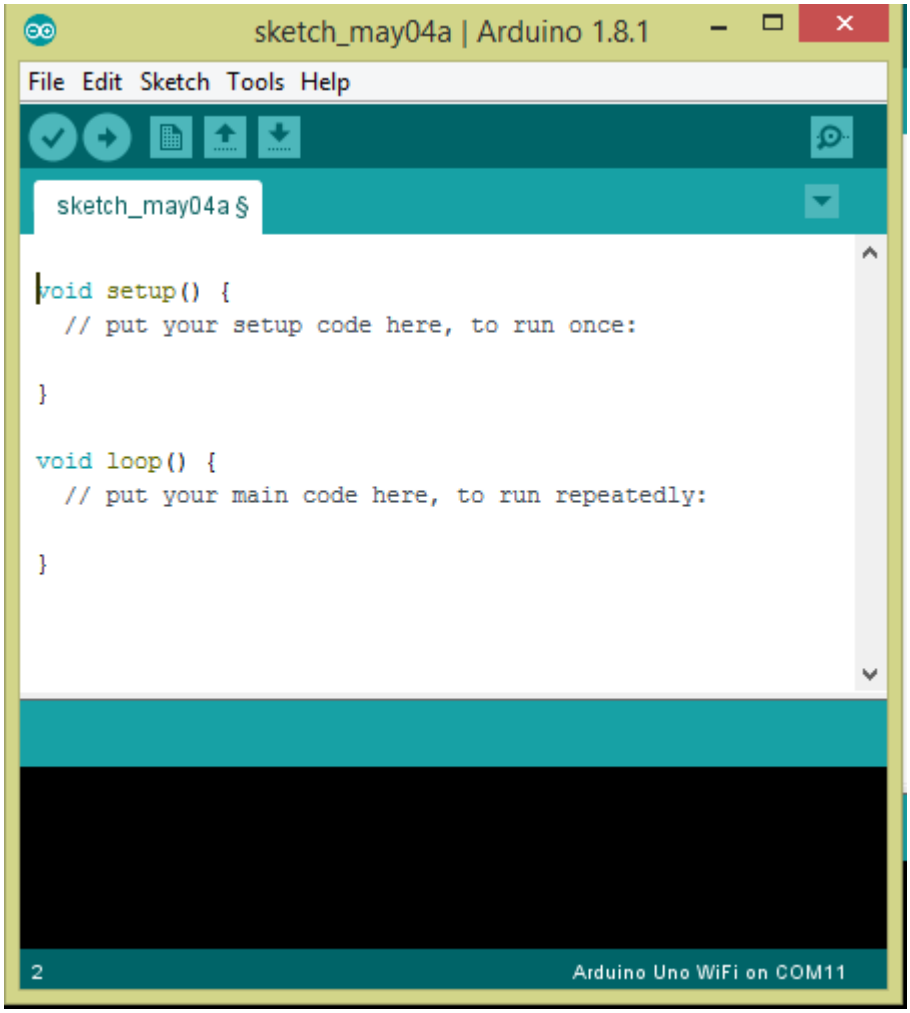
Joonis 13: Arduino Uno Wifi arendusplaat [20]

Seade koosneb trükkplaadist, millel on Atmel AVR mikrokontroller (ATmega 328), ESP8266 Wifi moodul, 14 digitaal sisend-väljund viiku ning 6 analoog sisendviiku, millega saab arendusplaati ühendada erinevate sensorite, LED-tuledega jne. Arendusplaadil on ka USB liides, millega ühendatakse seade arvutiga, et seadmele programme peale laadida ning andmeid kommunikatsiooni pordi kaudu edastada ja vastu võtta. Arendusplaat saab oma toite USB liidese kaudu või eraldi toitesisendist, kus soovitatav on 7-12 V alalispinge.

Arduinol on 3 erinevat mälu tüüpi- Flash mälu, SRAM (ingl. k static random access memory) ning EEPROM. Flash mälus (mälu tüüp, mis säilitab oma informatsiooni ka siis kui seade pole toideallikaga ühendatud) hoitakse Arduino programme, SRAM-is (mälu tüüp, mis pärast seadme toitest eemaldamist kaotab loodud informatsiooni) luuakse ja uuendatakse programmi muutujaid ning EEPROM on mälu, kuhu programmeerijad saavad salvestada informatsiooni ning peamiselt lisatakse sinna infot konfiguratsiooni kohta vms [21]. Nt. Cumulocity Arduino teek salvestab EEPROM-i seadme identifikatsiooni ning kui kasutaja üritab teist korda sama seadet registreerida, siis loeb ta EEPROM-ist selle välja.

Arendusplaadile programmide laadimiseks ja loomiseks kasutatakse Arduino arenduskeskkonda. Tegemist on Javas kirjutatud vabavaralise programmiga, mis töötab nii Windowsi, Linuxi kui ka Mac OS X keskkonnas.

Arduino IDE (Integrated Development Environment) võimaldab programme luua C++ keeles ning vajalik on defineerida vähemalt kaks funktsiooni - setup() ja loop() (joonis 14). Setup() käivitub programmi alguses ja seal seatakse erinevad algväärtused (nt. kas mingi viik on sisend- või väljundviik) ning luuakse algseid ühendusi (nt. GPRSi ühendus või seriaalühendus). Loop() käivitub selle järel. See on tsüklis ning tegemist on programmi põhifunktsiooniga.



Joonis 14: Arduino arenduskeskkond

GSM laiendusplaat

GSM ehk globaalne mobiilsidesüsteem (Global System for Mobile communications) on Euroopa standard, mis on saanud globaalseks mobiilse kommunikatsiooni standardiks. See sisaldas algselt endas 2G digitaal-mobiilsidevõrgu protokolle ning asendas kunagise analoog-mobiilsidevõrgu [22].

GSM laiendusplaadil on SIM900 GSM/GPRS moodul, mis toetab nelja erinevat sagedusala (850/900/1800/1900 MHz) [23]. Moodulit saab kontrollida AT käskluse abil. AT on lühend sõnast "ATtension" ning iga käsklus algab lühendiga "AT", millega antakse moodulile teada käsurea algusest. AT käsklustega saab näiteks infot GSM-mooduli kohta (nagu mudeli number, tootja nimi, tarkvara versioon jne), andmeid, sõnumeid ning kõnesid saata ja vastu võtta jne [24].

Antud laiendusplaadil on samuti antenn, SIM-kaardi lugeja, sisend-väljund viigud ning toitesisend, millele sobib 7-12 V alalispinge. SIM-kaart sisaldab infot telefoninumbri kohta ning hoiab piiratud arvu SMS-sõnumeid ja kontakte. Samuti on laiendusplaadil eraldi pistikud mikrofonile ja kõlarile.

GSM moodulit (joonis 15) kasutatakse seadme ja platvormi vahelise suhtluse läbiviimiseks läbi GPRS võrgu. GPRS (üldine raadio-pakettandmeside teenus) on andmeside teenus, mis on 2000-ndate aastate algusest kasutusel olnud ning on M2M (inglise keeles machine-to-machine) kommunikatsiooni saavutamiseks populaarne.

Joonis 15: GSM laiendusplaat [25]

Mooduli kasutamiseks on vajalik ka SIM-kaardi olemasolu, mille peaks panema SIM-kaardi hoidjasse. Soovituslik oleks samuti PIN-koodi eemaldamine eelnevalt. Võrku ühendumiseks oleks samuti vajalik teada oma teenusepakkuja pääsupunktinime.

Selleks, et GSM-moodul töötaks Cumulocityga, oleks vaja kasutada Cumulocity Arduino teeki, mis on kirjutatud C keeles ning samuti oleks vajalik Arduino arenduskeskkonda importida GSM-teek Open Electronics poolt. Cumulocity juhendis Arduino seadmetele soovitatakse ka nendes teekides muuta osasid faile vastalt viikidele, mida kasutati GSM-mooduliga kommunikatsiooni saavutamiseks [26].

GSM-teek võimaldab SMS-sõnumite saatmist, lugemist, kustutamist ning GSM-mooduli käivitamist ja käskluste saatmist. Samuti kõnede alustamist, vastuvõtmist ning katkestamist, GPS-i ühenduse loomist ning katkestamist ning seadme infot pärimist (koordinaate, kõrgust, UTC-aega jne). Cumulocity teegi jaoks on tähtsamad funktsionaalsused nagu HTTP-päringute saatmine, TCP ning GPRS ühenduse loomine ja katkestamine.

Cumulocity teek tegeleb Cumulocity platvormi mõõtmistulemuste saatmise, alarmide loomise, seadme registreerimise ning operatsioonide täitmisega. Enne on vajalik platvormile anda võimalus seadet autentida, mis on võimalik järgmise objektiga:

```
CumulocityPlatform(const char *host, const char *tenantId, const
char *user, const char *password, const char *applicationKey)
```

Antud objekt võtab argumentidena kasutaja hostinime (nt. “minupesa.cumulocity.com”), hostinime lokaalse osa (nt. “minupesa”), kasutajanime (nt. “minuemail@mail.com”), salasõna ja oma rakenduse võtme. Selle objekti väärtusi kasutatakse HTTP-päringute tegemisel, et API-liidesega suhelda.

CumulocityPlatform objektil on ka vaja GSM-mooduli objekti, et oleks mille kaudu saavutada ühendus Cumulocity platvormiga.

```
GSMModule* mod;
```

Setup() funktsioonis lähtestada mod nii:

```
mod = new GSMModule();
```

APN tuleks asendada oma teenusepakkuja pääsupunkti nimega ning tihti kasutajanimi ning salasõna puuduvad.

```
mod->attachGPRS("internet.emt.ee", "", "")
```

Kui GPRS-ühenduse loomine õnnestus, tuleks siduda GRPS-ühendus haldav muutuja Cumulocity teegi isendiga, et viimane kasutaks päringute saatmisel olemasolevat ühendust.

```
cPlatform.setGSM(mod);
```

Temperatuuri- ja niiskussensor

Antud digitaalne temperatuuri- ja niiskussensor (joonis 16) on Adafruiti AM2302, mis on nelja-viiguline versioon DHT22-st. Antud sensor sobib tavatingimustes mõõtmistulemuste võtmiseks - temperatuuri annab adekvaatseid tulemusi -40 - 80°C tingimuses ning niiskust loeb sensor 0-100% tingimuses. Uusi tulemusi võtab sensor umbes iga kahe sekundi tagant [27].

Joonis 16: Temperatuuri ja niiskuse sensor [27]

Sensor on integreeritud PCB-le (Printed Circuit Board), millega on viikude arvu ühe võrra vähendatud. Sensori üks viik on ühendatud arenduslaua 5 V viiguga, teine GND viiguga (0 V) ja kolmas digitaalviiguga.

Arduino arenduskeskkonnas andmete sisselugemiseks on võimalik kasutada cactus.io teeki, mis on DHT22-le mõeldud.

Kasutatud digitaalse viigu ära märkimine ning loome muutuja dht, mis saab kasutada AM2302 meetodeid :

```
#define AM2302_PIN 2
AM2302 dht(AM2302_PIN);
```

Arduino setup() meetodis sätitakse viik sisendi viiguks:

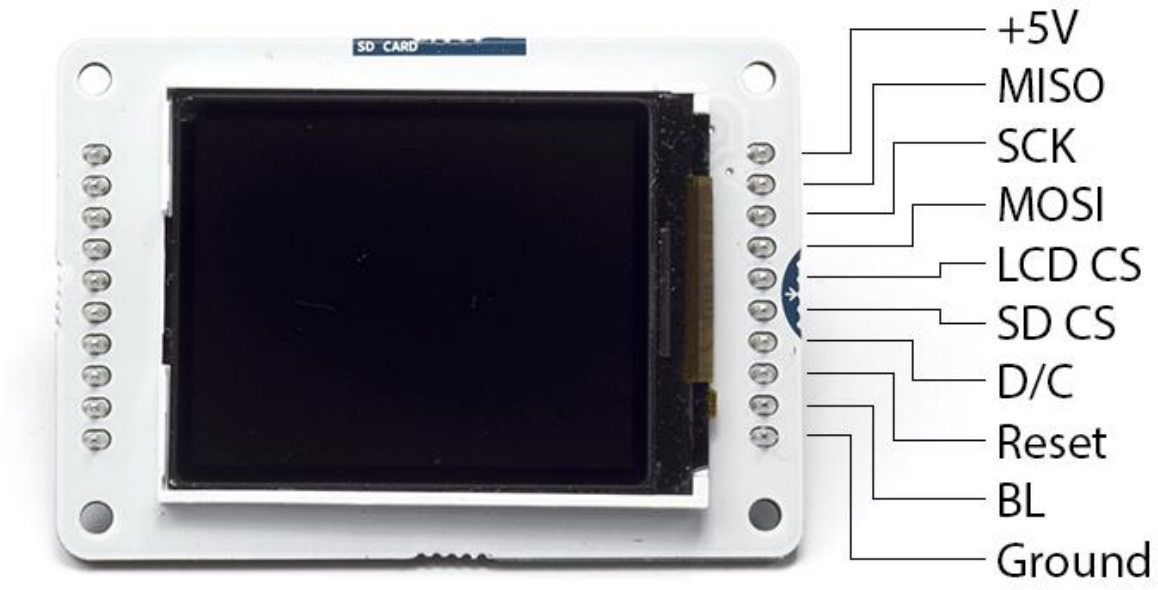
```
dht.begin();
```

Sellega on nüüd võimalik muutuja dht kaudu sensorilt andmeid küsida.

LCD ekraan

Arduino 1.77” TFT (ingl. k Thin-Film-Transistor) SPI (ingl k. Serial Peripheral Interface) LCD (ingl. k liquid-crystal display) on moodul, mille resolutsiooniks on 160x128 pikslit. Moodulil on ka micro SD-pesa, millele saab mälukaardi bitmap failidega lisada, et neid faile ekraanil kuvada. Antud seade nõuab 5 V toidet, mida saab pakkuda ka Arduino arendusplaat [28].

TFT LCD ekraan (joonis 17) on LCD ekraani tüüp, mis parandab LCD pildikvaliteeti. Igal pikslil on oma transistor, mis annab suurema kontrolli kasutatud värvide üle. Samas kuigi tulemuseks võib olla teravam pilt, on tulemus kõige parem otsevaates ning kõrvalt nurga alt vaadates võib olla ekraanil kujutatu raskesti nähtav. TFT LCD ekraanid nõuavad ka rohkem energiat, mis on ka üks põhjus, miks neid enam mobiiltelefonide ekraanidena ei kasutata [29].



Joonis 17: LCD ekraan [30]

SPI on liides, mis võimaldab andmeid vahetada kahe seadme vahel, kus üks on ülemseade (ingl. k *master*) ja teine alluvseade (ingl. k *slave*) või mitu alluvseadet. Alluvseadmeteks on välisseadmed nagu ekraan ja printer ning ülemseade mingi mikroprotsessor. Tegemist on täisdupleksi režiimis töötava standardiga, mis tähendab, et samal ajal saab andmeid nii saata kui vastu võtta. [31]

Joonisel 18 on näha, kuidas ülemseade ja alamseade on ühendatud nelja juhtmega, kus SCLK (ingl .k Serial Clock, samuti kasutatakse lühendit SCK) sünkroniseerib andmeedastust ja MOSI (ingl. k Master Out Slave In) on ühendus, millega ülem saadab andmeid alamseadmele ning MISO (ingl k. Master In Slave Out) ühendusega saadab alluv andmeid ülemale. SS (ingl. k Slave Select) on ühendus, millega ülem saab katkestada andmevahetuse alluvaga [31].

Joonis 18: SPI ülemseadme ja alluvseadme vaheline infovahetus [32]

Selleks, et antud LCD ekraani Arduinoga kontrollida, on vaja kasutada kahte teeki - Arduino TFT-teeki ja SPI-teeki. Esimest kasutatakse selleks, et ekraan käivitada ning nt. määrata, mis viike kasutati ning mis suuruses teksti ekraanile tahetakse kuvada ja määrata, mis tekst või pildi nimi on, mida tahetakse ekraanil näha. Teine tegeleb suhtlusega kahe seadme vahel.

TFT-teek põhineb kahel teegil- Adafruit GFXja Adafruit ST7735. Esimene neist tegeleb joonistamise tegevustega ning teine tegeleb Arduino ekraaniga. TFT-teek kasutab SPI-teeki ning seetõttu on vajalik see oma programmi lisada, isegi kui programmis ise pöördutakse otse vaid TFT-teegi poole [33].

Oma Arduino programmis peaks TFT-ekraani käivitamiseks kõigepealt märkima viigud, mida kasutati CS, DC ja RST ühenduste jaoks. Järgnevalt tuleks teha *TFTscreen* nimeline objekt, mis võtab defineeritud viikude väärtused enda argumentidena.

```
#define cs   10
#define dc   9
#define rst  5

TFT TFTscreen = TFT(cs, dc, rst);
```

Siis setup meetodis peaks isendile rakendama begin funktsiooni, mis alustaks suhtlust ekraani ja Arduino arenduslaua vahel.

```
TFTscreen.begin();
```

Sumisti

Passiivne sumisti (inglise keeles buzzer) on elektriline helisignaali allikas, mis kasutab ära piesoelektilist materjali (nt. piesokvartsi). See materjal deformeerub vahelduva elektrivälja mõjul ning tulemuseks on kuuldav toon. Sumistil (joonis 19) on plastmassist silindriline raam ning ülemisel osal auk heli levimiseks. Sumisti sees on ketas, mis on vajalik heli loomisel [34].

Joonis 19: Sumisti [34]

Arduino programm

Seadme registreerimine Cumulocity Arduino teegi kaudu

Seadme registreerimiseks on kasutatakse Cumulocity Arduino teegi funktsiooni signatuuriga:

```
int CumulocityPlatform::registerDevice(const char* name, char* buffer, int bufferSize)
```

Funktsioon registerDevice võtab endale 3 argumenti- seadme nimi, muutuja, kuhu salvestatakse seadme number Cumulocity platvormis ning mitu sümbolit mahub muutujasse.

```
char id[8];
cPlatform.registerDevice("arduino_uno", id, 8);
```

Mõõtmistulemuste kogumine ja saatmine

Kõigepealt loetakse uued mõõtmisväärtused sensorilt ning teek salvestab need väärtused temperature_C ja humidity muutujatesse. Mõõtmistulemused saab kätte kui salvestatakse need programmi muutujatesse.

```
dht.readHumidity();
dht.readTemperature();
float temperatuur = dht.temperature_C;
float niiskus = dht.humidity;
```

Mõõtmistulemuste saatmiseks tuleb kasutada funktsiooni sendMeasurement, mille signatuur on järgmine:

```
int CumulocityPlatform::sendMeasurement(const char* type, const char* fragmentName, const char* measurementName, const long &mValue, const char* mUnit)
```

Ehk klassil CumulocityPlatformil on funktsioon sendMeasurement, mis võtab argumentideks mõõtmistulemuse tüübi (nt. temperatuur või niiskus), kuidas Cumulocity seda nimetab, mõõtmistulemuse tüübi nimi, väärtus ning mõõtmistulemuse ühik. Funktsioon tagastab täisarvu, mille väärtuse järgi saab teada, kas mõõtmistulemuse saatmine oli edukas.

Temperatuuri tulemuse saatmine:

```
cPlatform.sendMeasurement("Temperature",
"c8y_TemperatureMeasurement", "T", temperatuur, "C");
```

Niiskuse tulemuse saatmine:

```
cPlatform.sendMeasurement("Humidity",
"c8y_HumidityMeasurement", "h", niiskus , "%RH");
```

Aja pärimine GSM-moodulilt

Kohaliku aja saamiseks oleks mõttekas GSM-moodulit kasutada. Selleks tuleks kasutada AT-käsklusi:

```
AT+CMGF=1
AT+CENG=3
AT+CCLK?
```

AT-käsklus AT+CMGF=1 määrab ära, et GSM-moodul peaks olema SMS teksti režiimis, kus SMS sõnumeid esitatakse loetava tekstina. AT+CHGF võib võtta oma väärtuseks ka nulli, millega on sõnumid esitatud binaarse stringina, mis on kodeeritud kuueteistkümnendsüsteemi sümbolitega [35].

AT-käsklus AT+CENG=3 aktiveerib režiimi, kus on võimalik näha detailset infot kasutatava võrgu kohta ning AT+CCLK? annab kohaliku kuupäeva ja aja, mida saab kasutada ekraanil kuvamiseks [36]. Ainult AT+CCLK? käskluse kasutamine võib anda vale tulemuse, kus näidatakse seda kui kaua on GSM-moodul on aktiivne olnud.

GSM-mooduliga suhtlemiseks on vaja kasutada GSM-teeki, kus käskluste saatmiseks kasutatakse järgmise signatuuriga funktsiooni:

```
void SIMCOM900::SimpleWriteln(const __FlashStringHelper
*pgmstr)
```

AT+CMGF=1 käsu saatmine moodulile käiks nii:

```
gsm.SimpleWriteln(F("AT+CMGF=1"))
```

GSM-mooduli vastuse sisselugemiseks oleks vajalik kasutada funktsiooni järgmise signatuuriga:

```
uint8_t SIMCOM900::read()
```

Tagastustüübiks on täisarv, mis on 8-bitine (ehk viimane arv on binaarkujul 11111111 ehk 255). Vaikimisi tagastab funktsioon read() väärtuse 255. Iga number tähistab tähte, mis lisatakse sõnale, et saada täielik GSM-mooduli vastus käsklusele.täisarvu muutmine täheks:

```
int a = gsm.read();
letter = (char)a;
```

Operatsioonide pärimine Cumulocitylt

Selleks, et Cumulocity platvormist küsida, kas on lõpetamata operatsioone, oleks vaja luua kaks muutujat (tähtede järjendid), kuhu saab salvestada leitud operatsiooni väärtused.

```
char operation[100];
char operationName[50];

cPlatform.getPendingOperationFromServer(operationName, 50,
operation, 100);
```

Funktsioon getPendingOperationFrom Server võtab oma argumentideks loodud muutujad, kuhu antud funktsioon lisab leitud operatsiooni nime ja sisu väärtustena. Samuti on vaja lisada kuni mitu tähte saab kummagisse muutujasse lisada. Nt. antud juhul võib olla operationName kuni 50 tähte pikk.

Arduino programm otsib kahte operatsiooni, millest üks annab teada, et tuleks saata mõõtmistulemuste info platvormi ning teine annab teada, et käivitada tuleks alarm. Alarmi loomine toimub Cumulocity keskkonnas üles pandud komponendi kaudu ning mõõtmistulemuste operatsioon tekib regulaarselt CEP-mooduli abil, mis asub joonisel 9.

Operatsiooni lõpetamine

Operatsiooni lõpetamiseks oleks vaja muuta operatsiooni staatus väärtuselt “PENDING” väärtusele “SUCCESSFUL”. Selleks kasutatakse funktsiooni järgmise signatuuriga:

```
int CumulocityPlatform::markOperationCompleted()
```

Antud funktsioon otsib üles operatsiooni identifitseeriva numbri ning kasutab seda, et muuta ära operatsiooni staatus vaid sellel operatsioonil. Funktsioon tagastab siin täisarvu, mille järgi on võimalik öelda, kas operatsiooni staatuse muutmine oli edukas.

Andmete kuvamine ekraanile

Tausta, teksti värvuse ja suuruse määramine:

```
TFTscreen.background(0, 0, 0);
TFTscreen.stroke(255, 255, 255);
TFTscreen.setTextSize(5);
```

Teksti kirjutamine koordinaatidel 5 ja 20:

```
TFTscreen.text(sensorPrintout, 5, 20);
```

Cumulocity komponentide loomine ja lisamine

Antud peatükis näidatakse täpsemalt, kuidas teha komponente alarmi loomise komponendi näitel ning samuti, kuidas kasutada Cumulocity enda komponente ja neid keskkonda juurde lisada.

Alarmi loomise komponent

Antud alarmi loomise komponent sisaldab endas välju, et sisestada kuupäeva ja kellaaega, millal peaks äratuskell reageerima. Siis saadetakse Arduinole operatsioon ajaliste andmetega ning see jäetakse meelde kuni tuleb aeg, mil alarm peaks aktiveeruma.

Esiteks peaks olema vajalik Node.js (version 6.7 või uuem) ja cumulocity-tools teek Node.js-ile alla laadimine. Kui Node.js on alla laaditud (nt. kodulehelt või käsurealt), siis cumulocity-tools teegi saab globaalselt oma masinale järgmise käsuga:

```
$ npm i cumulocity-tools -g
```

Järgnevalt peaks looma uue kausta, kuhu komponenti looma hakata. Siis tuleks genereerida package.json fail, mis sisaldab tähtsaid metaandmeid projekti kohta (nt. Projekti version ning määrata projekti sõltuvused erinevate teekide suhtes). Npm-i käsk selleks on järgmine:

```
$ npm init
```

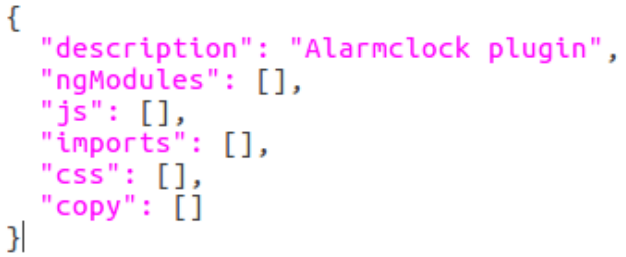
Kui see käsk on antud, siis küsitakse kasutajalt projekti kohta erinevaid andmeid nagu projekti nimi, kirjeldus, versiooninumber, autori nimi jne. Järgmine samm oleks alla laadida Cumulocity UI korpuse (inglise keeles package) viimane versioon.

```
$ c8y install latest
```

See käsk lisab Cumulociy UI ka faili package.json-isse sõltuvusena. Komponendi loomiseks on vajalik järgmine käsk, mis loob õige projektistruktuuri ja vajalikud failid:

```
$ c8y create:plugin alarmclock
```

See loob kausta “plugins”, kus on kaust “alarmclock” ning selles kaustas on genereeritud fail cumulocity.json (joonis 20). Samuti peaks sinna faili lisama nime, mida komponentide valikus kuvatakse.



Joonis 20: Genereeritud cumulocity.json fail

Järgnevalt tuleks luua fail alarmclock.module.js, mis näeks välja selline:

```
(function () {
  'use strict';

  angular.module('myapp.alarmclock', []);
}());
```

Järgmisena tuleb luua alarmclock.config.js fail, mille eesmärgiks on lisada asukoht komponendile. Kuna tegemist on komponendiga, siis tuleks lisada loodud moodul komponentide loetellu. Selleks kasutame C8y.UI mooduli objekti c8yComponentProvider. Objektil on meetod add, millega saame lisada uue komponendi vaatesse. Selleks loome funktsiooni configure, mis lisab JSON-i objekti vajalike andmetega komponentide loetellu. Nt. lisatud on komponendi nime (peab olema unikaalne väärtus), loetelus näidatava nime ja kirjelduse ning samuti html-faili relatiivne asukoht "Plugin" kaustas.

Siis tuleks teha kontrolleri fail "alarmclock.controller.js", kus on vaja täpsustada, milliseid mooduleid kasutatakse ning luua kontroller, mis sisaldab endas komponendi loogikat. Kuna üheks komponendi funktsionaalsuseks on saata Arduino seadmele operatsioon, siis on vaja kasutada Cumulocity moodulit "c8yDeviceBulkControl". Samuti tuleb defineerida ära skoobis erinevad muutujad, mida kasutatakse, sest kasutajaliideses on igale kuupäeva ja aja osale eraldi väli ning nende väärtused on vaja sisse lugeda.

Kui kasutaja on sisestanud andmed ning vajutanud nupule, siis kontrollitakse, kas tegemist on korrektse kuupäevaga ning kas soovitud aeg alarmi käivitamiseks on veel ees. Kuupäeva valiidsuse kontrollimiseks kasutati moment.js teeki järgmiselt:

```
moment($scope.year+"-"+$scope.month+'-'+$scope.day, 'YYYY-MMM-
DD').isValid();
```

Kui kuupäev ei olnud õige, siis tagastati vastav sõnum kasutajale ning kui kuupäev oli adekvaatne, siis uuriti, kas kell ja kuupäev olid tulevikus. Kui kuupäev ja kellaaeg olid sobilikud, siis luuakse operatsioon nii:

```
var operation = {groupId:"****", creationRamp:1,startDate:
resultDate, operationPrototype:{ description:"Scheduled
operation for Arduino", Clock_control:{}}}
```

```
c8yDeviceBulkControl.create(operation);
```

Ehk algselt sai loodud JSON-i objekt, mis sisaldas endas infot Arduino seadme kohta ning sisaldas tunnust “Clock_control”, mille Arduino üles leiab, sest operatsioon sisaldab sõna “control”, mida Arduino Cumulocity teek otsib. Loodi hulkoperatsioon, mis loob planeeritud operatsiooni seadme grupile, kus on äratuskella seade. CreationRamp viitab hilinemisele iga operatsiooni loomise järel enne järgmise operatsiooni loomist ning operationPrototype viitab operatsiooni infole – seal on kirjeldus ja fragment, mille Arduino seade tuvastab.

Kui need failid on tehtud, siis tuleks otsida üles kompondendi cumulocity.json fail ning seda täiendada järgnevalt:

```
{
  "description": "Plugin for alarm clock",
  "name":"Alarm Clock",
  "ngModules": ["myapp.alarmclock"],
  "js": [
"alarmclock.module.js","alarmclock.config.js","alarmclock.controller.js"
],
  "imports": [],
  "css": [],
  "copy": []
}
```

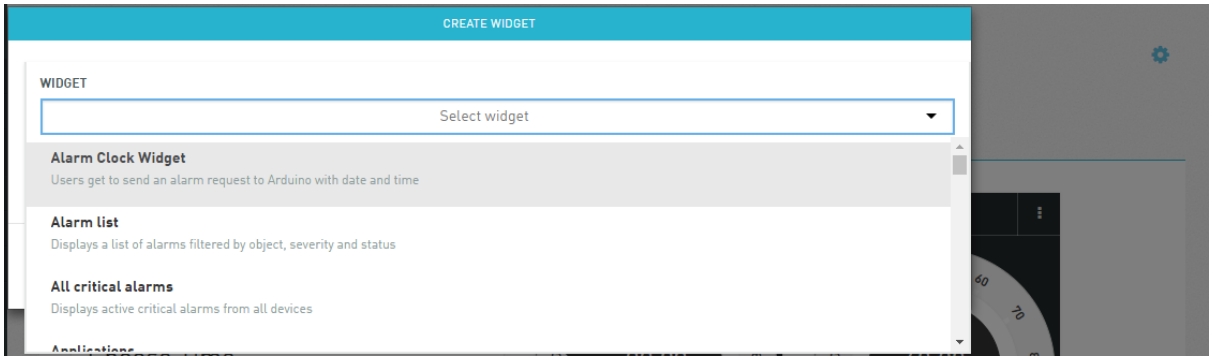
Joonis 21: Lõplik cumulocity.json fail komponendile

Selleks, et komponenti lisada rakendusse, on vajalik .zip fail genereerida, mis paneb kokku terve mooduli. Selleks saab kasutada käsureal cumulocity-toolsi nii:

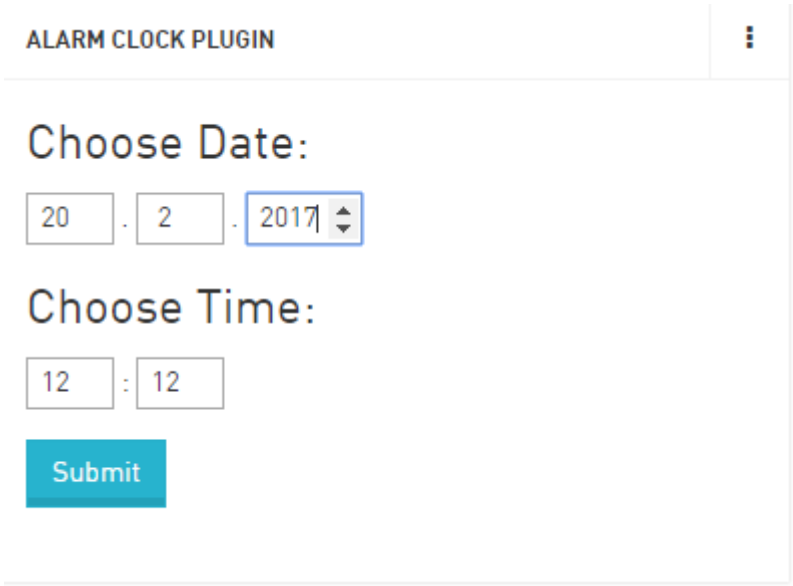
$ c8y build:plugin alarmclock väljundkaust

Antud käsk genereerib vajaliku .zip faili kausta väljundkaust, mis on kättesaadav projekti juurkaustas. Järgmisena peaks Cumulocity keskkonnast üles otsima rakenduse administratsiooni rakendusest ning selle komponentide akna lahti tegema ja zip faili lisama vajutades paremale kastile (joonis 12).

Siis kui avada rakendus ja minna esipaneeli vaatesse, peaks olema uus komponent nähtav komponentide loetelus (joonis 22). Komponent näeks välja nagu joonisel 23. Komponendi stiili ning pealkirja saab muuta kui komponenti lisada esipaneelile või kui võtta komponendi seaded lahti.



Joonis 22: Esipaneeli uue komponendi lisamine

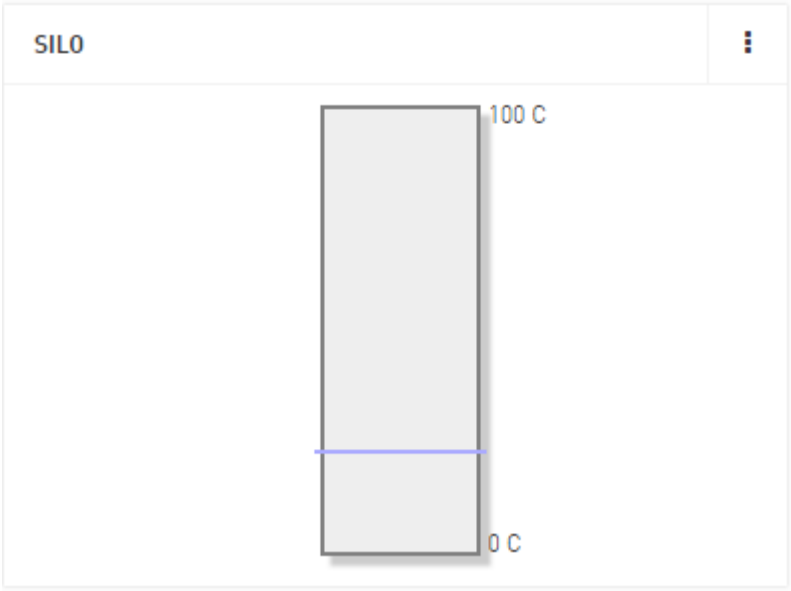


Joonis 23: Alarmi aja sisestamise komponendi vaade

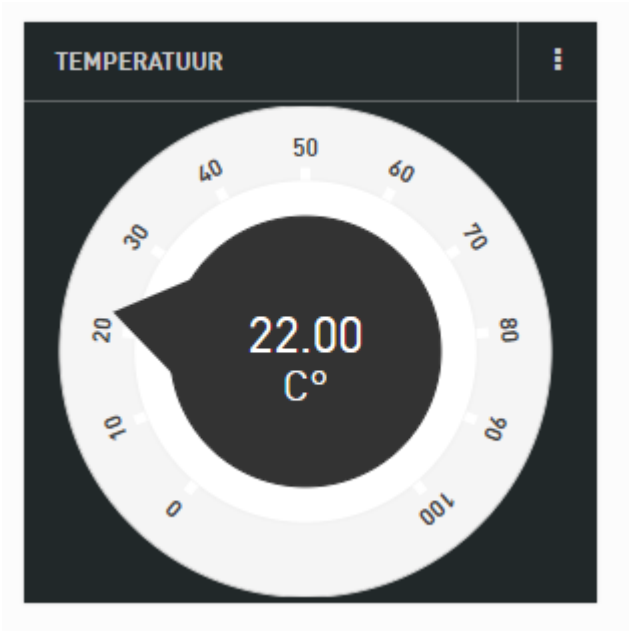
Cumulocity komponendid

Kuna Cumulocity keskkonnas on ka palju komponente, siis seadme esipaneelile peale alarmi komponendi lisaks ka viimaste mõõtmiste tulemused ning graafik, mis näitab nt. viimase päeva mõõtmisi.

Viimaste mõõtmistulemuste jaoks saab kasutada komponente “Silo” (joonis 24) ning “Radial Gauge” (joonis 25). Mõlemad on juba kättesaadavad esiplaani komponentide seast.

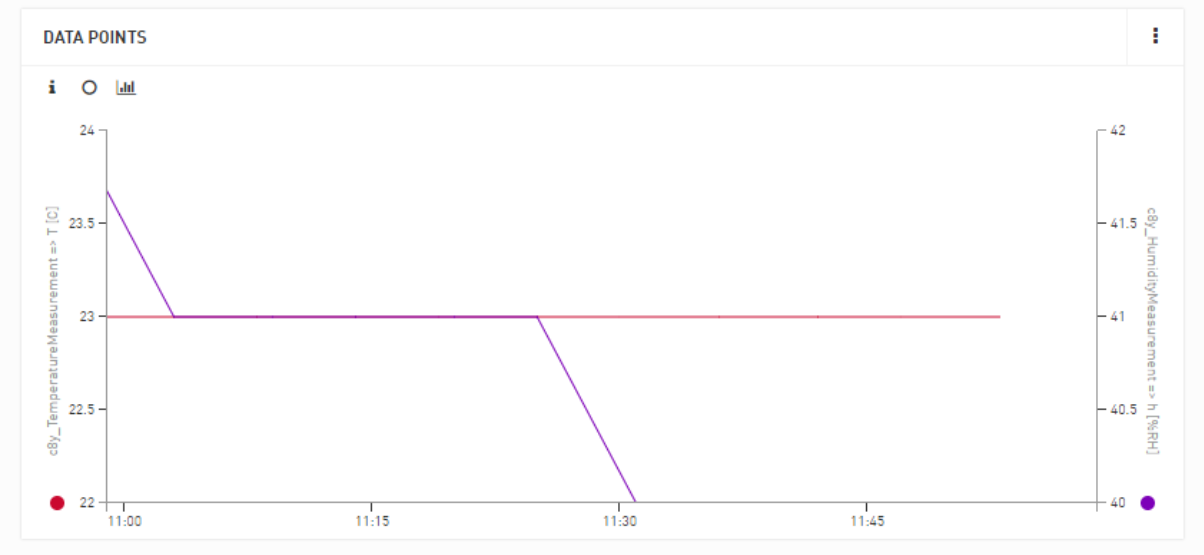


Joonis 24: Silo komponent



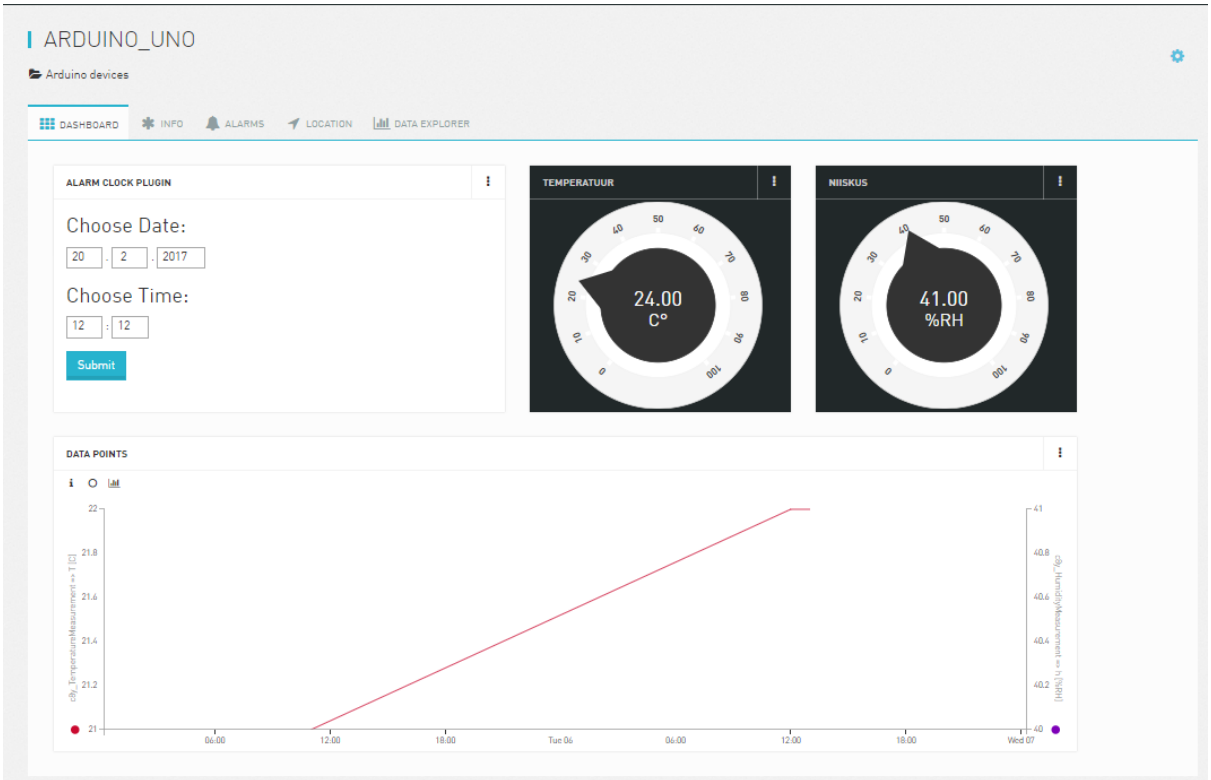
Joonis 25: Komponent Radial Gauge

Mitmekülgse graafiku saab kui valida pluginate menüüst “Data points graph”. Siis saab valida ajaperioodi, mõõtmistulemuste tüübi ning kas tegemist on reaalajas end uuendava graafikuga. Siis saab esipaneelile sarnase graafiku, mis on joonisel 26.



Joonis 26: Mõõtmistulemuste graafik esipaneelil

Lõpptulemus komponentide lisamisel tühjale esipaneelil on joonisel 27.

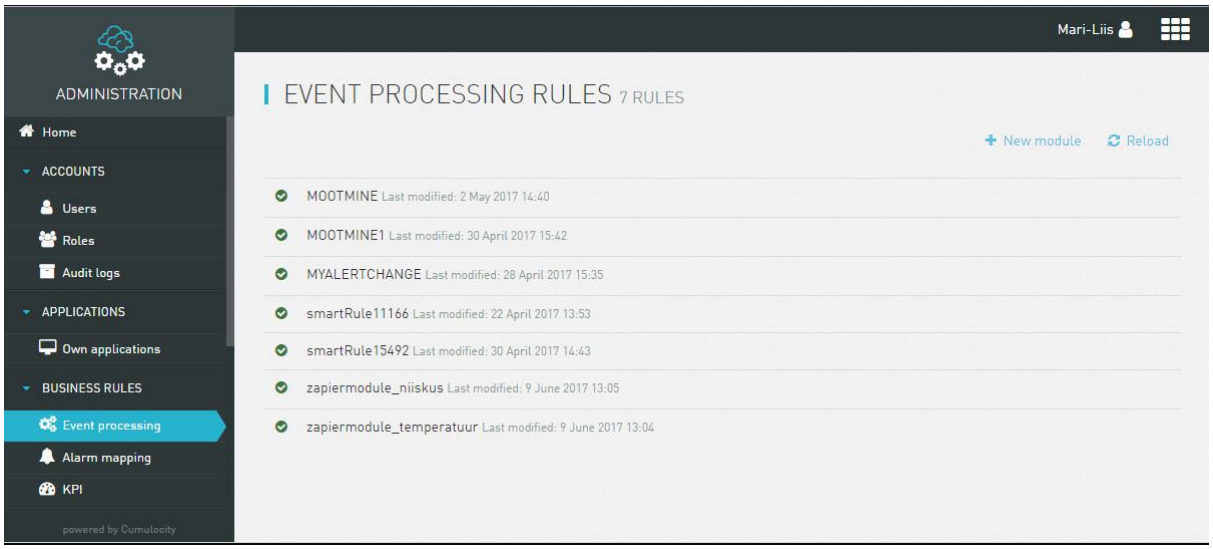


Joonis 27: Ekraanitõmmis esipaneeli vaatest

Zapieri kasutamine

Kuna Cumulocity säilitab andmeid limiteeritud perioodiks, siis andmete säilitamiseks on mõttekas need viia nt. kusagile Google Spreadsheeti.

Esiteks oleks vaja luua CEP moodulid temperatuurile ja niiskusele. Neid peaks eraldi võtma, sest nende andmestruktuur ei ole sama ning ei saaks mõõtmistulemuse väärtust ega ühikut kätte. CEP-mooduleid saab luua nii Postmani kaudu kui ka Cumulocity keskkonnas adminstratsioonirakenduses "Event Processing" alt (joonis 28).



Joonis 28: CEP-moodulid kuvatuna administratsioonirakenduse vaates

Näide temperatuuri andmestruktuurist mõõtmistulemuste saatmisel platvormile JSON-I vormingus:

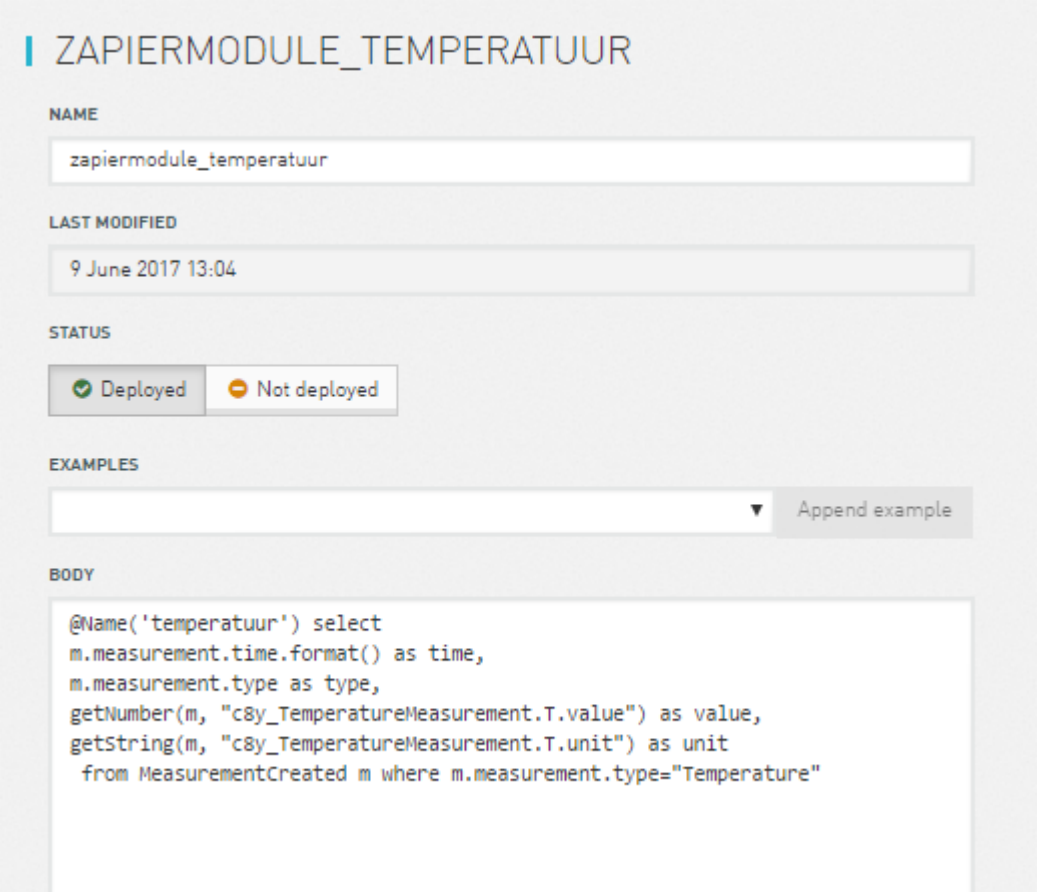
```
{"source":{"id":"10370"},
  "time":"2017-06-09T09:04:12.527Z",
  "type":"Temperature",
  "c8y_TemperatureMeasurement":{"T":{"value":23,"unit":"C"}}
}
```

Näide niiskuse andmestruktuurist mõõtmistulemuste saatmisel platvormile JSON-I vormingus:

```
{"source":{"id":"10370"},
  "time":"2017-06-09T09:04:27.919Z",
  "type":"Humidity",
  "c8y_HumidityMeasurement":{"h":{"value":41,"unit":"%RH"}}
}
```

Zapieri ühenduse tegemine temperatuuri andmete jaoks

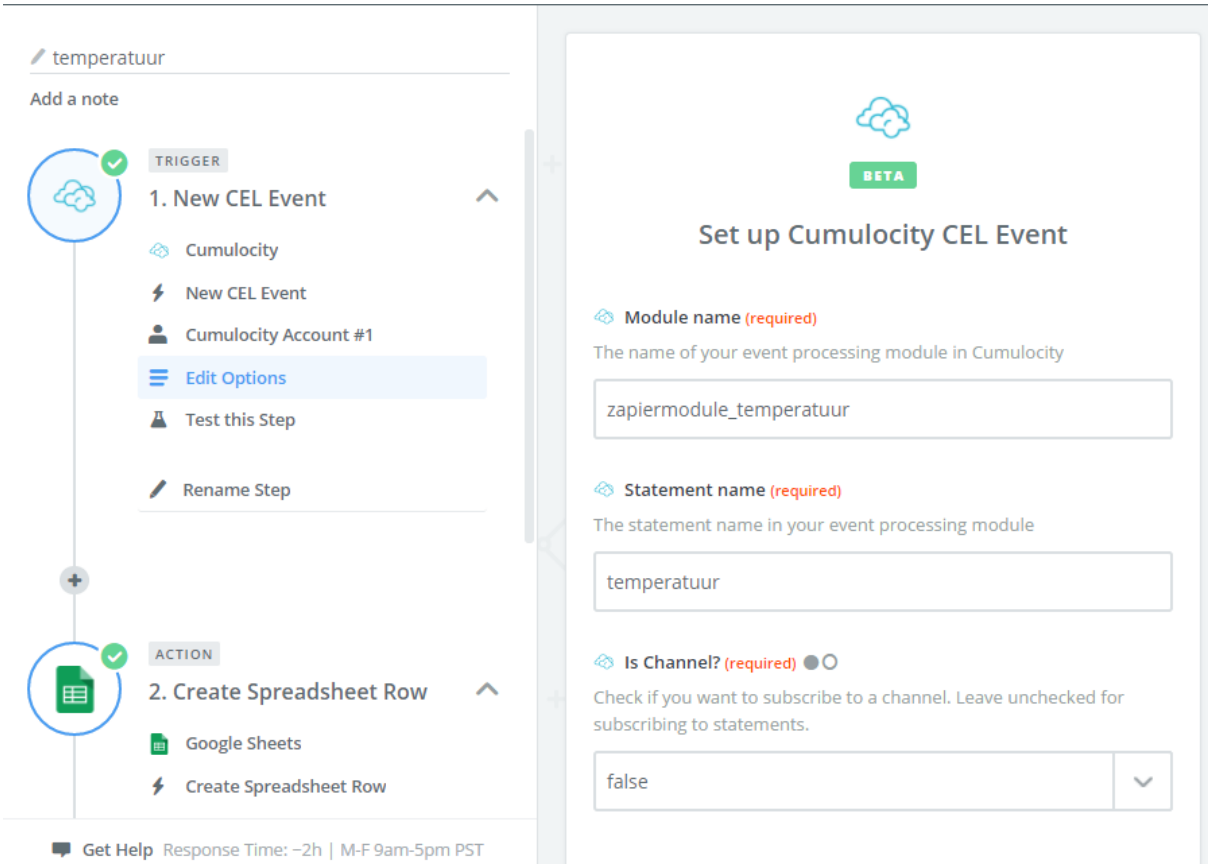
Alustama peaks Zapieri mooduli loomisest ning selleks tuleks kasutada voogu MeasurementCreated, kuhu lähevad kõik loodud mõõtmistulemused. Mooduli loogika on selline, et kui loodud mõõtmine on temperatuuri tüüpi, siis Zapierile edastatakse selle mõõtmise aeg, tüüp, väärtus ning ühik. Töötav moodul on joonisel 29.



Joonis 29: Zapieri moodul temperatuurile

Järgmine samm oleks Zapieris ühenduse loomine Cumulocity ja Google Spreadsheetsi vahel. Selleks tuleks teha uus Zap, millega seome CEP mooduli ja Spreadsheedi.

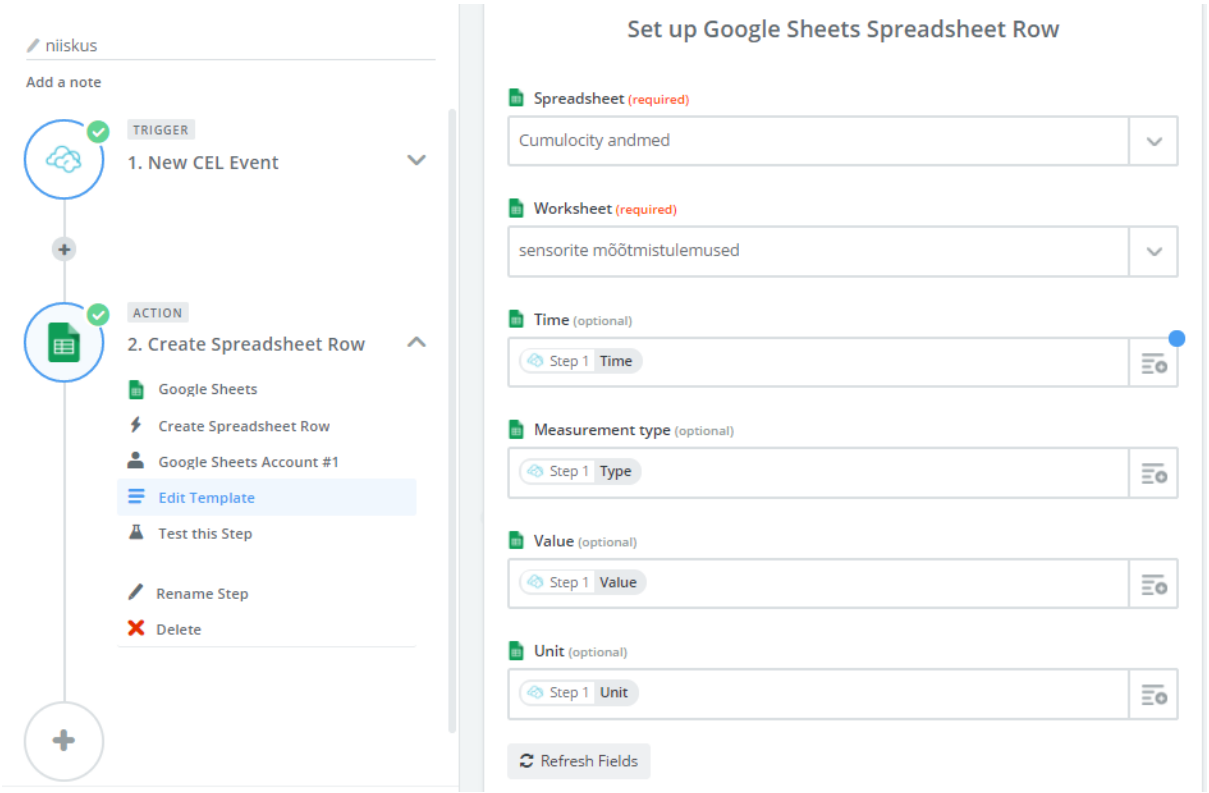
Triger



Joonis 30: Zapieri vaade CEP-moodul sidumisel

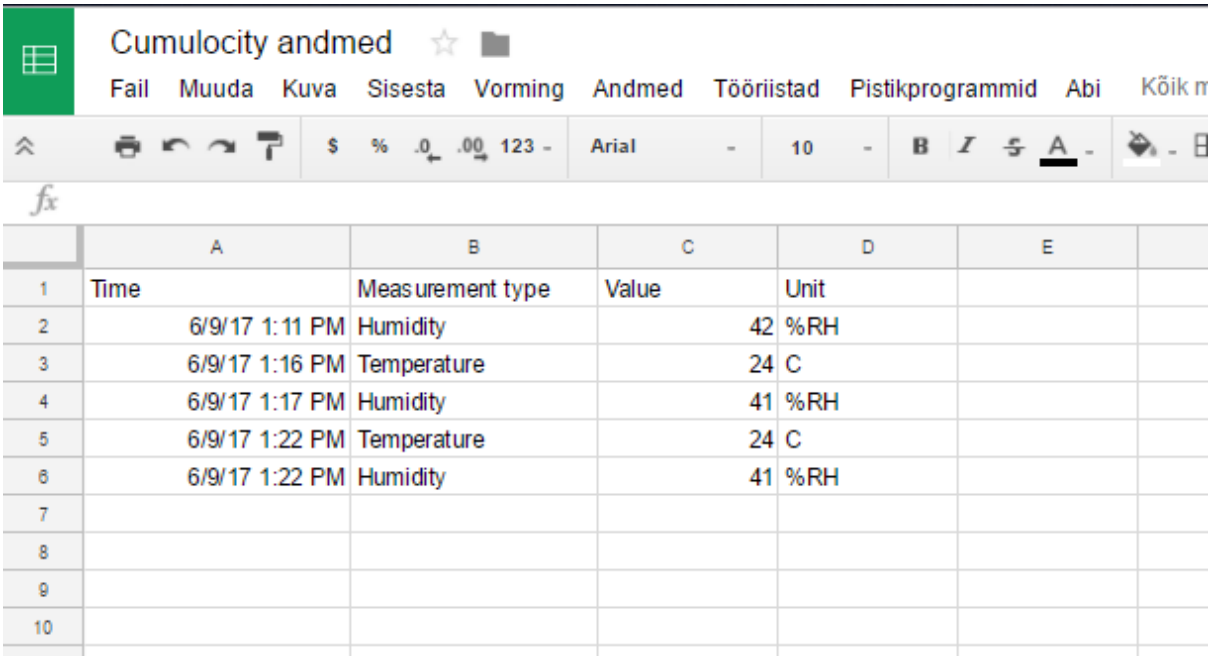
Enne tuleks siduda ka Cumulocity kasutaja Zapieriga ning otsustada ära, kas triger on CEP-moodul või tark reegel. Trigeriks on loodud CEP-moodul ning vaja on anda selle mooduli ja lause nimi (joonisel 29 on näidatud loodud CEP-moodulile vastavad andmed).

Tegevus



Joonis 31: Google Spreadsheeti real lisamise mall

Tegevuse osa kirjeldab seda, mida pärast trigerit tegema peaks. Esiteks tuleks saada ühendus oma Google Spreadsheets kasutajaga ning siis otsustada, kuhu faili ja lehele andmeid saata. Järgmisena tuleks valida lisatakse uus rida või muudetakse mingit rida (antud näitel luuakse uus rida). Spreadsheetis on algselt täidetud neli välja, kuhu on lisatud väärtused Time, Type, Value ja Unit. Zapier sisestab uued väärtused esimesse tühja ritta nende veergude alla (tulemus joonisel 32).



	A	B	C	D	E
1	Time	Measurement type	Value	Unit	
2	6/9/17 1:11 PM	Humidity	42	%RH	
3	6/9/17 1:16 PM	Temperature	24	C	
4	6/9/17 1:17 PM	Humidity	41	%RH	
5	6/9/17 1:22 PM	Temperature	24	C	
6	6/9/17 1:22 PM	Humidity	41	%RH	
7					
8					
9					
10					

Joonis 32: Google Spreadsheeti faili vaade

Kokkuvõte

Käesoleva bakalaureusetöö eesmärgiks oli anda ülevaade sellest, kuidas Cumulocity platvormi kasutada ning luua Cumulocity platvormiga ühendatud sensoritega äratuskell, mis kuvaks ekraanil kohaliku aja ning temperatuuri sensorilt saadud mõõtmistulemuse ning mis suhtleks Cumulocity platvormiga - saadaks mõõtmistulemusi regulaarselt oma Cumulocity keskkonda ning varundaks need ka Google Spreadsheet lehele kasutades CEP-mooduleid ning Zapieri vahendajana, võtaks vastu uusi aegu äratuskellale alarmide loomiseks läbi autori loodud komponendi.

Lõputöö raames valmis Arduinol põhinev äratuskella prototüüp, mis oli ühendatud erinevate sensorite ja moodulitega. Äratuskell kasutab GSM-moodulit, et saada kohalik aeg ning teha päringuid Cumulocity API-le, et saata mõõtmisandmeid ning uurida, kas äratuskellale on saabunud uus alarm, mida tulevikus käivitada. Cumulocity keskkonda tehti esipaneel, kuhu on lisatud komponendid, mis kuvavad viimatisi mõõtmistulemusi, reaalajaline graafiku komponent, mis kuvas mõõtmistulemusi mingi perioodi vältel ning komponent, mis võimaldas kasutajal lisada uusi aegu, millal alarm käivitada äratuskellal.

Cumulocity platvormi osas peab nentima, et tegemist on väga võimeka ja paindliku platvormiga, millel on palju erinevaid tööriistu, mis teevad arendajate ja süsteemi haldajate töö lihtsamaks IoT süsteemide loomisel ja administreerimisel. Samuti peaks mainima, et tegemist on pidevat areneva platvormiga ning umbes aastaga, mil autor on uurinud ja katsetanud Cumulocity platvormi, on see palju kasutussõbralikumaks muutunud. Kui oleks vaja luua mingit komplekssemat süsteemi, siis suurema tõenäosusega oleks selle platvormi pakutav lisandväärtus suurem kui autori loodud näite puhul.

Enne bakalaureusetöö kirjutamist oli autoril vaid pinnapealne varasem kokkupuude mikrokontrolleritega ning riistvara programmeerimisega. Cumulocity platvormiga varasem kogemus puudus lõputöö autoril ning tegemist oli autori meelest kõige keerulisema osaga aru saada, kuidas Cumulocity keskkonda kasutada ning selles orienteeruda.

Kasutatud kirjandus

[1] Nest, Nesti temperatuuri kirjeldus, https://nest.com/thermostat/meet-nest-thermostat/ (8.06.2017)

[2] Chris Poulin, Security Intelligence, „The importance of IPv6 and the Internet of Things", https://securityintelligence.com/the-importance-of-ipv6-and-the-internet-of-things/ (8.06.2017)

[3] Ted Navarro, Compute Next, „Distributed cloud computing will be the essential part of the Internet of Things", https://www.computenext.com/blog/distributed-cloud-computing-will-be-an-essential-part-of-the-internet-of-things (8.06.2017)

[4] McKinsey Global Institute, „Unlocking the potential of the Internet of Things", http://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/the-internet-of-things-the-value-of-digitizing-the-physical-world (8.06.2017)

[5] Cumulocity, „Introduction to Cumulocity", https://cumulocity.com/guides/concepts/introduction/ (8.06.2017)

[6] High-Tech Gründerfonds, „HTGF and VersoVentures invest in Cumulocity GmbH – The M2M Spin off of Nokia Siemens Networks (NSN)", http://high-tech-gruenderfonds.de/en/htgf-and-versoventures-invest-in-cumulocity-gmbh-the-m2m-spin-off-of-nokia-siemens-networks-nsn/ (8.06.2017)

[7] „Introducing JSON", http://www.json.org/ (8.06.2017)

[8] Tutorialspoint, „RESTful Web Services - Introduction", https://www.tutorialspoint.com/restful/restful_introduction.htm (8.06.2017

[9] W3schools, „HTTP Methods: GET vs POST", https://www.w3schools.com/tags/ref_httpmethods.asp (8.06.2017)

[10] Cumulocity, „Real-time processing", http://cumulocity.com/guides/concepts/realtime/ (8.06.2017)

[11] Zapieri keskkond, https://zapier.com/ (8.06.2017)

[12] Cumulocity, „SaaS Integration", http://www.cumulocity.com/guides/users-guide/saas-integration/ (8.06.2017)

[13] Cumulocity, „Web SDK for plugins", https://www.cumulocity.com/guides/web/introduction/ (8.06.2017)

[14] Npm, „Cumulocity-tools", https://www.npmjs.com/package/cumulocity-tools (8.06.2017)

[15] Wikipedia, "AngularJS", https://en.wikipedia.org/wiki/AngularJS (8.06.2017)

[16] Mari-Liis Kaldvee, "FoxMinded/collection", https://github.com/FoxMinded/collection

[17] Arduino, „Arduino UNO Wifi", http://www.arduino.org/products/boards/arduino-uno-wifi (8.06.2017)

[18] Arduino, „Software", https://www.arduino.cc/en/Main/Software (8.06.2017)

[19] Wikipedia, „Arduino", https://en.wikipedia.org/wiki/Arduino (8.06.2017)

[20] Arduino, „Arduino Uno Wifi",http://static.arduino.org/media/k2/galleries/214/A000133-Arduino-Uno-wifi-1front.jpg (8.06.2017)

[21] Arduino, „Memory", .https://www.arduino.cc/en/tutorial/memory (8.06.2017)

[22] Wikipedia, „GSM", https://en.wikipedia.org/wiki/GSM (8.06.2017)

[23] Simcom, „SIM900", http://simcom.ee/modules/gsm-gprs/sim900/ (8.06.2017)

[24] Syeda Anila Nusrat, Code Project, „Introduction to AT commands and its uses", https://www.codeproject.com/Articles/85636/Introduction-to-AT-commands-and-its-uses (8.06.2017)

[25] „SIM900 GSM-module”, https://ae01.alicdn.com/kf/HTB1h8yYNXXXXXaCaXXXq6xXFXXXx/Smart-Electronics-SIM900-GPRS-font-b-GSM-b-font-font-b-Shield-b-font-Development-Board.jpg (8.06.2017)
[26] Cumulocity, „Arduino”, http://www.cumulocity.com/guides/devices/arduino/ (8.06.2017)
[27] Adafruit, „AM2302 (wired DHT22) temperature-humidity sensor”, https://www.adafruit.com/product/393 (8.06.2017)
[28] Arduino, „GTFT”, https://www.arduino.cc/en/Main/GTFT (8.06.2017)
[29] Margaret Rouse, Tech Target, „Serial Peripheral Interface (SPI)”, http://whatis.techtarget.com/definition/serial-peripheral-interface-SPI (8.06.2017)
[30] Arduino, „GLCD pins”, https://www.arduino.cc/en/uploads/Main/GLCD_pins.png (8.06.2017)
[31] Arduino, „SPI”, https://www.arduino.cc/en/reference/SPI (8.06.2017)
[32] Wikipedia, „SPI single slave”, https://et.wikipedia.org/wiki/Kasutaja:Z3n/SPI#/media/File:SPI_single_slave.svg
[33] Arduino, „TFT”, .https://www.arduino.cc/en/Guide/TFT (8.06.2017)
[34] Engineers Garage, „Insight - How piezo buzzer works”, https://www.engineersgarage.com/insight/how-piezo-buzzer-works (8.06.2017)
[35] Diafaan, „AT CMGF”, https://www.diafaan.com/sms-tutorials/gsm-modem-tutorial/at-cmgf/ (8.06.2017)
[36] Importgeek, „Obtain time from GSM network using SIM900”, https://importgeek.wordpress.com/2016/01/13/obtain-time-from-gsm-netwok-using-sim900/ (8.06.2017)

Lisad

Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks
Mina, Mari-Liis Kaldvee,
1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose 1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

Cumulocity platform,
mille juhendajad on Anne Villems, Alo Peets ja Taavi Duvin,
2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tallinnas, 09.11.2017