TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Tarkvarateaduse Instituut

Kaarel Allemann 142410IABB

KODUAUTOMAATIKA TARKVARA ARENDAMINE TEENUSEPÕHISEST ÄRIMUDELIST TULENEVAID NÕUDEID JÄRGIDES

Bakalaurusetöö

Juhendaja: Enn Õunapuu

Doktorikraad

Tallinn 2017

Autorideklaratsioon

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

Autor: Kaarel Allemann

19.05.2017

Annotatsioon

Antud lõputöö eesmärk oli luua koduautomaatika tarkvara, mis võimalikult efektiivselt toetaks teenuspõhist ärimudelit. Õnnestumise hindamiseks koostati teenuse kirjelduse põhjal ülesanded, mida tarkvaraarendus pidi täitma, et toetada teenust võimalikult suurel määral. Kogutud ülesannete põhjal defineeriti tarkvaraarenduse põhimõtted ning loodi neljast komponendist koosnev koduautomaatika süsteem. Kõrvutades tulemit teenuse poolt püstitatud ülesannetega järeldati, et tarkvara ja selle arendamise protsess omavad teenuses püstitatud funktsionaalsust, on modulaarselt laiendatavad, vajavad minimaalset seadistamist ja suudavad automatiseeritult uueneda. Ainsa ülesandena ebaõnnestus taristu loomine kasutuskogemuse arvestamiseks arendusprotsessis, mis analüüsi tulemusel osutus arhitektuuri valikust tulenenud probleemiks. Töö käigus loodud tarkvara koos arendusmetoodikaga on kasutusel järgmistes teenusearendusetappides.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 42 leheküljel, 8 peatükki, 10 joonist.

Abstract

Developing home automation software based on requirements by service-based business model

The goal of this thesis was to create a home automation software, which would adequately follow requirements set by service-based business model. In order to estimate results, the list of requirements based on service description was compiled for software development. Using these goals, software development process was put in place, which produced four-part home automation software. Comparing these results to service requirements shows that software and its development process support required functionality, are modular, require minimal configuration and could automatically update. Only drawback was an effort to build infrastructure, in order to utilize user feedback in development process. Analysis showed that this was due to the choice choice of system architecture made in design process. Software and processes created in this thesis will be further used in service developing process.

The thesis is in Estonian and contains 42 pages of text, 8 chapters, 10 figures.

Lühendite ja mõistete sõnastik

Angular2	Mobiili ja võrgurakenduste arendusplatvorm [1]
API	Application Programming Interface- reeglistik programmiga suhtlemiseks [2]
Arduino Nano	ATmega328 protsessoril põhinev mikroarvuti [3]
Continuous integration	Praktika, mis automatiseerib koodibaasi täienemisel vajalikud protsessid (nt testimine) [4]
Ionic 2	Mobiilirakenduste loomiseks mõeldud raamistik, mis loomisel on kasutatud Angulari ja Apache Cordova [5]
Logi fail	Fail, kuhu salvestatakse programmi poolt märgitud sündmused [6]
MVC	Model-View-Controller- Arhitektuuri muster kasutajaliidese realiseerimiseks [7]
MySql	Relatsiooniliste andmebaaside haldamise süsteem [8]
Node.js	Javascripti käitussüsteem [9]
Orange Pi Zero	Mikroarvuti, mis kasutab ARM tüüpi protsessorit ja on võimeline kasutama Linuxil põhinevaid operatsioonisüsteeme [10]
ORM	Object-relational mapping- Programmeerimistehnika objektorienteeritud keeltele, mis loob nö virtuaalse objektida andmebaasi, mida saab keele siseselt kasutada [11]
PostgreSql	Relatsiooniliste andmebaaside haldamise süsteem [12]
QR-kood	Quick Response Code- Masinloetav maatriksi põhine vöötkood [13]
React	Javascripti teek kasutajaliideste ehitamiseks [14]
React Native	Mobiilirakenduste loomiseks mõeldud raamistik, mis võimaldab rakenduste arendamiseks kasutada Reacti ja Javascripti [15]
Raspberry Pi	Ühendkuningriigis arendatud mikroarvutite seeria, mis põhineb ARM protsessoril [16]
Sails.js	Populaarne MVC mustrit rakendav Node.js raamistik [17]
Teek	„Kollektsioon funktsioone, makrosid, klasse vms komponente, mis on mõeldud korduvkasutuseks programmides“ [18]

Sisukord

Jooniste loetelu

1 Sissejuhatus

Üks olulisemaid suundi infotehnoloogiste lahenduste arengus on olnud üleminek tootepõhilistelt ärimudelitelt teenuspõhilistele mudelitele. See protsess ei piirdu pelgalt kliendi hinnastamisviiside muutmisega, vaid eeldab laiemaid muutusi tarkvaraarenduses ja ka lahenduse funktsionaalsuses. Tulenevalt erinevast ärilisest taustast võivad valikud ja metoodikad, mis viisid eduka toote arenduseni, tunduvalt erineda neist, mis viivad eduka teenuseni. Samal ajal on koduautomaatika valdkond, mis ei ole vaatamata riistvara mõõtmete ja hindade vähenemisest suutnud murda inimeste igapäevaellu teiste tarbeesemetega võrreldaval tasemel. Ühe probleemina võib näha vähest teenuspõhist lähenemist.

Käesolev töö võtab aluseks konkreetse koduautomaatika teenuse kirjelduse poolt esitatavad nõuded ning seab eesmärgiks luua neid järgiva tarkvara ja selle arendusprotsessi. Põhiküsimuseks on seega, millist mõju omavad tarkvaraarenduse käigus tehtud valikud tarkvara ja teenuse vahelisele kooskõlale. Kõrvutades tarkvara loomise käigus tehtud valikuid teenuse poolt seatud nõuetega, saab anda analüütilise hinnangu antud töös valitud lähenemistele. Nii on võimalik paremini mõista tarkvaraarenduse ja teenuspõhiliste mudelite vahelist suhet.

Töö koosneb kokku kuuest sisupeatükist. Teine peatükk kirjeldab koduautomaatika teenust ning toob välja, milliseid ülesandeid see püstitab. Kolmas peatükk avab laiemalt koduautomaatika tarkvarade tausta uurides sarnaseid tarkvarasid. Neljas kuni seitsmes peatükk katavad ära lõputöö käigus langetatud valikud ja nende tulemused, sisaldades infot tarkvaratehnikast, tehnoloogiatest ning teostamisest. Viimane ehk kaheksas peatükk hindab, kuidas lahendus vastab algselt püstitatud eesmärkidele, analüüsib saadud tulemust ning toob välja edasised arengusuunad teenuse ja valminud lahenduse suurema kooskõla saavutamiseks.

2 Tarkvaraarenduse eesmärgid

Antud töö põhifookuses on teenuse poolt püsitatud eesmärkide realiseerumise hindamine tarkvaras ja selle arendusprotsessis. Mõistmaks, millised on teenuse poolt seatavad nõuded tarkvarale ja milliste kontrollitavate eesmärkidena need väljenduvad, tuleb uurida, millistel mudelitel põhineb aluseks olev teenus ning milline võiks olla selle teenuse kirjeldus.

2.1 Koduautomaatika teenus

Koduautomaatika erakliendile turustamine teenusena koosneb kahest peamisest ülesandest. Esiteks peab olema teenusele tagatud tugi tark- ja riistvara hooldusena. Teiseks peab teenuse turustamise mudel erakliendile olema sarnane teiste juba turul tegutsevate pakkujatega, nagu voogedastusteenused, ajalehed ja ajakirjad, kuutasulised toitlustusteenused. Nende eesmärkide katmiseks kasutab koduautomaatika teenus sümbioosi tellimuspõhise ja hallatud teenuse mudelitest.

2.1.1 Teenuse mudelid

Tellimuspõhise teenuse puhul maksavad kliendid perioodiliselt, kusjuures teenuse hind ei sõltu kasutuse mahtudest [19]. Tihti pakutakse tasulist teenust lisana mõnele tasuta teenusele [19]. Viimasel ajal on tellimuspõhiste teenuste turule jõuliselt sisenenud digitaalmeedia pakkujad, nagu Netflix ja Spotify, kelle peamine innovatsioon ongi lähtunud ärimudeli ümberkujundamist [20]. Sarnasuse saavutamiseks ning turule paremini sobitumiseks on ka koduautomaatika teenuse eesmärk jääda viimastega sarnasesse hinnaklassi. Tellimuspõhise mudeli probleemina võib näha vähest suhtlust teenuse osutaja ja kliendi vahel, selle puuduse korvamiseks rakendab koduautomaatika teenus hallatud teenuse mudelit.

Hallatud teenus on teenuseosutamise liik, kus teenuse pakkuja hooldab ja jälgib regulaarset enda poolt pakutavat teenust. Selline teenuse liik on suures osas suunatud äriklientide turule.

Hallatud teenustele on omane [21], et:

- Teenuse hulka kuulub pakutava süsteemi töökorras hoidmine ja vigade jälgimine
- Teenuse osutaja suudab sooritada oma põhilised ärioperatsioonid üle võrgu
- Riistvara renditakse ning see jääb pakutavas teenuses tagaplaanile

Hallatud teenusega sarnaselt peab koduautomaatika teenus olema võimeline kliendile pakkuma süsteemi korrashoidu ning uuendusi.

2.1.2 Teenuse kirjeldus

Koduautomaatika teenuse eesmärk on pakkuda tellimuspõhiselt kliendile töötavat ja uuenevat asjade interneti ökosüsteemi, mis toetab kliendi poolt vabalt valitud hulka seadmeid. Esimeses faasis toetab selline keskkond kolme tüüpi kasutust: relee- ja kasvuhoonejuhtimine ning ilmainfo kogumine. Teenus võimaldab kliendil neid seadmeid kontrollida nii sisevõrgust kui ka internetist, viimane pole suurema paindlikkuse võimaldamiseks siiski lahenduse kasutamise eelduseks.

Taristu osas toetub lahendus kõikide seadmete puhul internetivõrkudele. Selline lähenemine on eelistatud, sest süsteemi ei looda lisakeerukust ebavajalike võrguprotokollide näol ja seda toetav infrastruktuur on väga laialt levinud, mis võimaldab teenuse osutajalt säästa riistvara kuludelt. Lisaks muutub tarkvaraarenduse seisukohalt lihtsamaks pakkuda pidevaid uuendusi kõikidesse seadmetesse.

Keskne komponent koduautomaatika teenuse osutamisel on riistvara, millel lahendused opereerima peavad. Riistvara võib jagada üldjoones kolme kategooriasse: andurid ja releed, kontrollerid ja keskseade. Hinnast tulenevalt on selge, et riistvaraliselt on eelistatud odavamad ja madala kvaliteediga lahendused, sest kliendile seadmeid rentides on oluline hoida ühekordsed kulutused madalal, et vältida suurte investeeringute vajadust ning võimalikke likviidsusprobleeme.

Tarkvara seisukohalt on tähtsad komponendid kontrollerid ja keskseade, sest nendel toimub süsteemi jaoks vajalike lahenduste rakendamine. Mõlema keskmes on mikroarvuti Orange Pi Zero, see omab ühenduseks vajalikku Wi-Fi ja ka Etherneti võrgukaarti. Võime jooksutada Linuxi-põhiseid operatsioonisüsteeme lubab Orange Pi'l kasutada väga laia hulka erinevaid tehnoloogiad. Riistvaraliselt võib kontrollerite puhul olla vajalik ka analoogsignaali lugemine anduritelt. Kuna Orange Pi Zerol selline võimekus puudub, kasutatakse selleks Arduino Nano't, mis on võimeline sellist signaali lugema ja selle Zerosse edasi toimetama.

2.1.3 Teenusest tulenevad eesmärgid

Eelnevas teenuse kirjelduses tõusevad märksõnadena esile töötav ja uuenev ehk teisisõnu peab tarkvara toetama võimalikult suures mahus tellimuspõhise ning hallatud teenuse vahelist sümbioosi. Nende põhimõtete võimalikult efektiivseks täitmiseks saab tarkvarasüsteemile tuletada viis eesmärki:

- omab teenuse kirjelduses püsitatud funktsionaalsust
- on riistvaraliselt modulaarselt laiendatav
- vajab lõppkasutajalt minimaalselt konfigureerimist
- suudab automatiseeritult uueneda
- arvestab kasutajakogemust arendusportsessis

Väljatoodud punktide hilisem kontroll võimaldab tuvastada, millisel hulgal pakub tarkvara tuge teenusele ning mil määraval panustavad töös langetatud valikud nimetatud eesmärkide täitmisele.

3 Sarnased tarkvarad

Arendatava tarkvara tausta avamiseks ja olemasolevate lahenduste mõistmiseks on kasutatud võrdlusi sarnaseid eesmärke püstitavate tarkvaradega. Kuna antud töö käigus arendav lahendus on suhteliselt piiritletud teenuse poolt seatavate nõuete poolt, annab selline võrdlus kõige vajalikumat infot, mis võib õnnestumisele kaasa aidata.

Tarkvarade võrdlusesse valimisel arvestati kahe tingimusega. Esiteks peab tarkvara kasutama töös kirjeldatud teenusega võrreldavat arhitektuuri ja riistvaralist platvormi. Teiseks peab eksisteerima info, mille alusel saab põgusalt tutvuda konkreetsete tarkvarade tööpõhimõtetega. Selliste tingimuste kasutamine annab suurema võimaluse kasutada sarnaste tarkvarade kogemust antud töö käigus. Valituks osutusid järgnevad tarkvarasüsteemid:

3.1 openHAB

OpenHABi [22] näol on tegemist tarkvaraga, mis on keskendunud erinevate seadmete integreerimisele, pakkudes väga erinevatele toodetele ühest reeglite ja kasutajaliidese süsteemi. Projekt on avatud lähtekoodiga ning kogukonna arendusele toetuv [22], mis tänu suurele arendajate hulgale suudab pakkuda laia toodete spektrit. Sellise võimekuse pakkumise keerukusest tulenevalt on tegemist peamiselt tehnikateadlikule inimesele suunatud tarkvaraga, kes suudab hallata tekstipõhiseid konfiguratsioone ning parandada süsteemis tõrkeid kasutades logifaile [22].

OpenHABi filosoofia peamiseks osaks on, et kogukonda kasutades suudetakse pakkuda tuge maksimaalsele hulgale seadmetele ning selliselt üles ehitatud lahendust kasutades ei pea liikmed jääma lootma ühe firma innovatsioonile ja edule [22]. Kasutajale on üles ehitatud ka andmehaldus, kasutajale jääb otsustada, kas ja kuidas peavad andmed sisevõrgust lahkuma [22]. Kokku koosneb openHAB tarkvarasüsteem keskserverist, mobiilirakendustest ja soovi korral rakendatavast välisvõrgu serverist, millele lisaks töötavad süsteemis hallatavad seadmed [22].

3.2 Pimatic

Ka pimaticu [23] näol on tegemist vabavaralise koduautomaatika tarkvaraga, mis pakub suure hulga seadmete haldamiseks veebipõhist kasutajaliidest, reegleid ning tarkvara APIt. Riistvaraliselt on pimatic loodud töötama Raspberry Pi seadmetel, kus töötab Node.js server, mis teenindab veebilehte ning juhib kontrollitavaid seadmeid. Ühenduse väljapoole sisevõrku peab samuti tagama sama server, mis tähendab portide avamist ruuteris ning SSL-ühenduse jaoks vajalikku eelkonfigureerimise sooritamist [23].

Nii nagu openHABi puhul on võetud ka pimaticu arendamisel suund, kus kasutatakse kogukonda, et saavutada võimalikult laia toetuspind erinevate tootjate seadmete hulgas ning iga konkreetse seadme korrektne ühendamine süsteemi on kasutaja ülesanne. Sellest tulenevalt ei ole antud tarkvara näol kohe töötava lahendusega ning süsteemi paigalduse ja halduse peab sooritama tehnilist kompetentsi omav inimene.

3.3 ImperiHome

ImperiHome [24] on tarkvara Androidi ja Apple nutiseadmetele, mis töötab koduautmaatika süsteemideülese juhttarkvarana. ImperiHome on saadaval kahes erinevas versioonis - piirangutega tasuta versioonis ning tasulises versioonis. Lisaks võimalusele kontrollida kodus leiduvaid süsteeme, sisaldab äpp ka kaht APId - esimene võimaldab tootearendajatel ehitada toodetesse ImperiHome'i tugi, teine on väikene server, mida kasutades on võimalik luua täiendavat kontrollfunktsionaalsust [24].

Sisuliselt erineb ImperiHome eelnevalt kirjeldatud lahendustest peamiselt arhitektuuriliselt. Abstraktsiooni tase on viidud kõrgemaks ning lisaks seadmete otsesele juhtimisele toetatakse ka tervete süsteemide juhtimist [24], seejuures on sisevõrgu väliseks kasutamiseks vajalik ruuteris avada kontrollitavate süsteemide kasutatavad pordid. Kuna tegu on kommertstarkvaraga, siis on pööratud suuremat tähelepanu võimalikult lihtsale süsteemipaigaldamisele ning suur osa tehnilist keerukust on jäetud kas tootede arendajate või ImperiHome'i enda arendajate õlgadele. Nii on saadud tarkvara, mis küll toetab vähem seadmeid, kuid on tavakasutajale kergem rakendada.

3.4 Home Assistant

Home Assistant [25] on Pythonil põhinev vabavaraline lahendus, mis nagu ka pimatic, on arendatud silmas pidades Raspberry Pi platvormi. Sarnaselt eelnevate vabavaraliste lahendustega on tegu erinevaid seadmeid ühendava kontrollkeskkonnaga, mida ilmestavad kogukonna poolt arendatav riistvaraline tugi ning seadmete keeruline ühendamine kasutades tekstipõhist konfiguratsiooni [25]. Sisult pakub Home Assistanti platvorm kasutajale kasutajaliidest seadmete kontrollimiseks ning võimalust luua tekstipõhist automatiseerimisloogikat [25].

Arhitektuurilt sarnaneb Home Assistant samuti juba eelnevalt kirjeldatud tarkvaradele, kus lahendust realiseeriv server asub sisevõrgus ning välisvõrgust ligipääsuvõimaluse loomiseks on vajalik ruuteris edasi suunata serveri kasutatavad pordid [25]. Mööndusega võib öelda, et antud tarkvara omab ka nutiseadmete tuge, kuhu küll ei ole loodud platvormipõhist rakendust, vaid on piirdutud veebirakenduse eessüsteemi mobiilisõbralikuks kohandamisega.

3.5 Võrdluse tulemid

Kõige üldisemalt joonistub võrreldavate tarkvarade puhul välja trend, et mida lihtsamaks on muudetud seadmete ühendamine süsteemi ning süsteemi haldamine tavakasutajale, seda vähem omab süsteem ka funktsionaalsust. Võib küll öelda, et võrreldavad süsteemid on eri arengujärgus. Kuid samas on põhimõtteline erinevus, kas keskendutakse laiale funktsionaalsusele või lihtsale kasutuskogemusele ning see paistab välja ka võrdses arengujärgus olevate süsteemide (nt openHUB ja ImperiHome) puhul.

Ka paistavad kõik võrdluses kasutatud tarkvarad silma suhteliselt sarnase tarkvaralise arhitektuuriga. Esiteks on valitud seadmete ülene lähenemine, kus põhifookus on ühtse keskkonna pakkumine erinevatele toodetele. Teiseks kasutavad kõik lahendused välisvõrguga info vahetamiseks portide avamist. Mõlema valiku taga võib peapõhjusena välja tuua erinevuse tasulise ja tasuta tarkvara vahel. Raske olekski näha, miks või kuidas peaks kogukonna toel ülesehitatud lahendus hakkama tootma oma seadmeid, samas on selline valik loogiline tasulise tarkvara puhul. Sama kehtib ka portide avamise puhul, mis tehnikateadlikule kogukonnale on tunduvalt lihtsam lahendus kui ühe lisa tarkvarakomponendi kasutuselevõtt. Samal ajal ei saa aga tasuline tarkvara eeldada kasutajalt keerulist tehnilist kompetentsi.

Üldistatult võib öelda, et peamised erinevused tarkvarade vahel tulenevad kasutajaskonna oskustest ning hinnast, mis määravad, milliseid põhimõttelisi ja arhitektuurseid valikuid peavad arendajad langetama. Antud töö tarkvarale tähendab see, et vähese funktsionaalsuse tõttu on võimalik ja tulebki muuta konkreetsete seadmete haldamine tavakasutajale võimalikult lihtsaks ning et välisvõrgust ligipääsetavuse probleem peab olema lahendatud tarkvaras, et mitte lisada kasutajale täiendavat keerukust.

4 Tarkvaratehnika

Koduautomaatika teenuse mudelite kirjeldusest lähtuvalt on tellimuspõhise hallatud teenuse pakkumine kahepoolne partnerlus. Ühelt poolt tagab klient oma tellimusega teenuseosutajale finantsile kindluse, teisalt peab teenuseosutaja hoidma süsteemi töökorras ning muutuvate nõuetega keskkonnas ka pidevalt funktsionaalsust täiendama. Seetõttu on oluline, et mitte teenus ise, vaid ka selle loomise viis, oleksid vastavuses teenuse osutamise vajadustega. Kasutatav tarkvaratehnika peab toetama partnerlust kliendiga ja hoidma pakutava teenuse kvaliteedi kõrgel.

Tarkvaraarenduse seisukohalt väljendab endas samu põhimõtteid - töötav tarkvara, reageerimine muutustele, koostöö klientidega- agiilne arendus [26]. Põhimõtete kokkulangevusest tulenevalt on loogiline, et antud töö tulemusena valminud süsteemi arendamise puhul langes rõhk just agiilsete praktikate kasutamisele. Järgnevalt on toodud kirjeldus tarkvaratehnika komponentidest, millele pöörati enim tähelepanu.

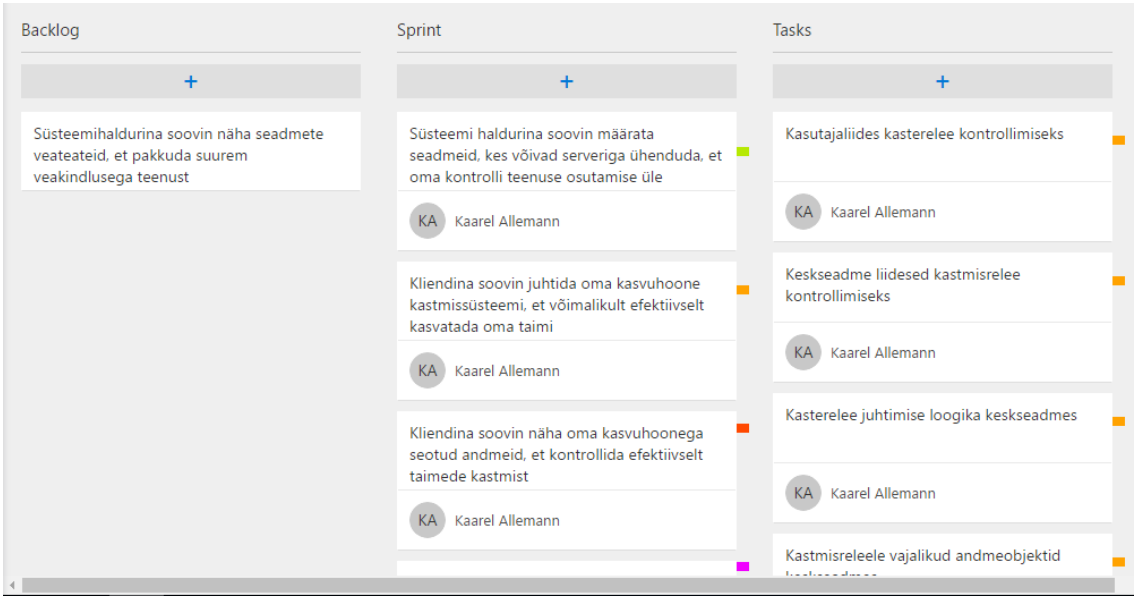
4.1 Scrum

Scrum on kergesti rakendatav agiilne projektiarendusmetoodika, mille peamised põhimõtted on [27]:

- Töö käib ajaliselt piiritletud iteratsioonides ehk sprintides
- Kogu tehtava vajamineva funktsionaalsuse kohta on logi, kust valitakse osa realiseerimseks sprindis
- Sprindi käigus realiseeritakse ainult alguses kokkulepitud funktsionaalsus
- Sprindi käigus realiseeritud tarkvara peab töötama
- Peamine infovahetus toimub igapäevasel koosolekul
- Kõige olulisemad tööle rakenduvad nõuded on ajalised

Üks Scrumi peamiseid eeliseid on, et metoodika ei suru peale suurt hulka dokumentatsiooni ega muid suuri administratiivseid kohustusi [27]. See muudab Scrumi hästi kasutavaks erinevate suurustega meeskondade korral. Konkreetse töö kontekstis sai see määravaks põhjuseks, miks valiti just see metoodika. Kuna töö autor oli ühtlasi ka ainuke süsteemi arendaja, võimaldas Scrum metoodika suunata põhirõhu

arendustegevusele, kuid samas säilis ka piisaval hulgal infot arendusprotsessi kohta, mis võimaldab selle tunduvat kergemat liikumist mõne suurema meeskonna kätte.



Joonis 1 Scrumi rakendamine

Antud töös kasutati Scrumi rakendamiseks keskkonda nimega Microsoft Planner (vt Joonis 1). Seal on töö jagatud kolme tulpa: vasakul projekti logi, keskel käimasolev sprint ja paremas, käimasoleva sprindi ülesanded. Projekti eripärast tulenevalt loobuti tavaliste praktikate, nagu igapäevased koosolekud ja ülesannete hindamise kasutamisest. Need praktikad loovad väärtust peamiselt arendajate vahelise infovahetuse elavdamisega, kuid ühe inimese arendatavas lahenduses pole sellised praktikad vajalikud ning nende lisamine tulevikus ei sega olemasolevat protsessi.

4.2 Kasutajalood

Eduka tarkvaraarenduse üks olulisemaid eeldusi on õigete nõuete mõistetavalt kirjapanemine. Eelnevalt toodud Scrumi kirjelduses on näha projekti logi, kust valitakse välja osad, mida realiseerida sprindis. See tähendab, et peab olema viis, kuidas mõistlikult tükeldada nii logi kui ka sprinte, et realiseeritavad funktsionaalsed nõuded oleksid võimalikult lihtsalt ja üheselt mõistetavad. Üks võimalik lahendus nõuete selliseks jagamiseks on kasutada kasutajalugusid [27].

Kasutajaloo võib defineerida kui lühikese kavatsuse, mis kirjeldab, mida süsteem peab kasutaja jaoks tegema [27]. Kasutajalugude vorm koosneb kolmest peamisest komponendist: roll, tegevus ja väärtus [27]. Neid kasutades võtavad laused vormi: *rollina*

soovin *tegevus*, et *väärtus* [27]. Näiteks võib kasutajalugu olla: „Kliendina soovin kontrollida releesid, et juhtida elektriseadmeid“.

Koduautomaatika süsteemi arendamisel langes valik kasutada kasutajalugusid nõuete defineerimisel, sest võrreldes näiteks alternatiivsete kasutusjuhtudega on nende dokumenteerimine tunduvalt lihtsam. Keeruliste diagrammide asemel saab panna sprindis sisalduvad nõuded kirja paari kindlas vormis moodustatud lausega. Olulist rolli mängis ka, et peametoodikana on kasutusel Scrum ning kasutajalood jagunevad kergesti sprindisiseselt mugavalt kasutatavateks kindlateks tööülesanneteks.

Nagu näha jooniselt 1, on nii logis kui ka käimasolevas sprindis kasutatud funktsionaalsete nõuete realiseerimiseks kasutajalugusid. Sprindis olevate kasutajalugudele on määratud täitja ning ka värv. Sama värvi kasutavad ka konkreetse kasutajaloo alla käivad ülesanded, nii on võimalik arendajal kiirelt haarata, mis kasutajaloo funktsionaalsust realiseeritakse.

4.3 *Continuous deployment* (CD)

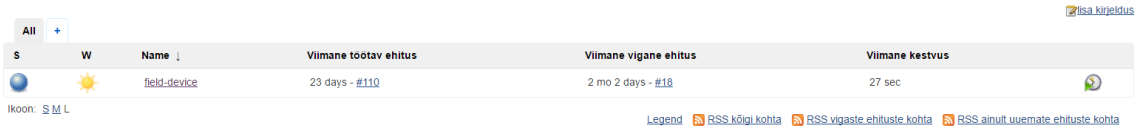
Üks otseselt agiilsetest põhimõtetest tulenev praktiline lahendus on pidev uue funktsionaalsuse pakkumine kliendile. Sellise praktilise lahenduse realiseerimist tarkvaraarenduse protsessi osana võibki nimetada CDks. Sisuliselt on CD *continuous integration*’i edasiarendus, kus lisaks ühtsele koodibaasile ja testimisele, on automatiseeritud ka välja töötatud lahenduste rakendamine süsteemis [28]. CD rakendamise peamisteks kasuteguriteks on lahenduste kiirem turule jõudmine, tarkvara kõrgem kvaliteet ja töökindlus ning pidev tagasiside kasutajatelt [28].

CD koosneb kahest komponendist - esiteks kliendile perioodiliselt uue funktsionaalsuse pakkumisest ning teiseks võimekusest juhtida tarkvaraarendust kasutusinfost tuleneva pideva tagasisidevoo alusel [29]. Seda kasutades on võimalik hoida tarkvara pidevalt vastavuses klientide nõuetega ja hallata neile osutatavat teenust. CD edukaks ja tõrgeteta rakendamiseks tarkvaraarendusprotsessis saab välja tuua kolm peamist eeltingimust [29]

- Agiilsete praktikate rakendamine organisatsioonis
- Osadeks jagatud ja testitav koodibaas
- Protsessid klientidelt saadavad tagasiside kogumiseks ja kasutamiseks

Laiadest eeltingimustest tulenevalt nõuab CD praktika täielik rakendamine olulist hulka otseselt mittevajaliku tööd. Tulenevalt antud töö käigus arendatud lahenduse väiksusest

ja varajasest faasist, võinuks see tähendada ebapraktilist fookuse nihkumist arenduselt endalt. Samas oleks olnud teenuse eripärast tulenevalt ka tarbetu jätta CD realiseerimine täielikult tulevikku, kus see tähendaks iga juba kliendil kasutuses oleva seadme tarkvara manuaalset uuendamist. Seetõttu piirdub lahenduses CD rakendamine ainult teenuse osutamisel kliendi valduses olevate seadmetega.



Joonis 2 Projekt Jenkinsis

CD rakendamiseks on igasse kliendi valduses olevasse Linuxit kasutavasse seadmesse installeeritud programm nimega Jenkins. Jenkins võimaldab luua projekti (vt joonis 2), kus saab seadistada GitHubi repositooriumisse *pollimi*se, see tähendab, et seade teeb pidevalt päringuid repositooriumile ning kui midagi on muutunud, tõmbab uuendused alla. Jenkinsi ligipääs repositooriumile on lahendatud rakendusvõtmega, mis annab Jenkinsile allalaadimisõigused. Repositooriumid, mis on seotud seadmetega, on spetsiaalselt loodud tarkvarauuenduste pakkumiseks ning sinna üleslaetav kood on muudetud inimesele loetamatuks (vt joonis 3), et kaitsta arendatud tarkvara õigusi. Samuti võimaldab Jenkins automatiseerida tarkvaraliste sõltuvuste uuendamist ja vältida olukorda, kus klient peab jääma kasutama vananenud tehnoloogial põhinevaid lahendusi.

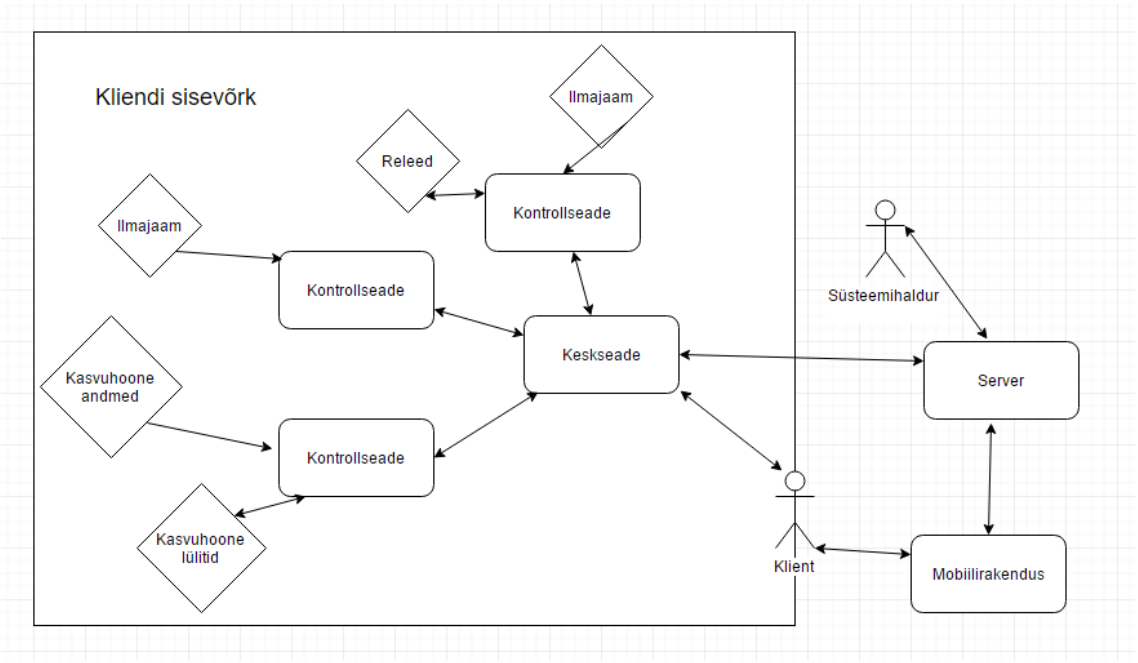
```
!function(e){function t(r){if(n[r])return n[r].exports;var o=n[r]={exports:{},id:r,loaded:!1};return e[r].call(o.exports,o,o.exports,t),o.loaded=!0,o.exports}var n={};return
t.m=e,t.c=n,t.p="/",t(0)}(function(e){for(var t in e)if(Object.prototype.hasOwnProperty.call(e,t))switch(typeof e[t]){case"function":break;case"object":e[t]=function(t){var
n=t.slice(1),r=e[t[0]];return function(e,t,o){r.apply(this,[e,t,o].concat(n))}}(e[t]);break;default:e[t]=e[e[t]]}return e}([function(e,t,n){n(809),e.exports=n(409)},function(e,t,n){"use
strict";e.exports=n(53)},function(e,t,n){(function(e){!function(t,n){e.exports=n()}(this,function(){"use strict";function t(){return vr.apply(null,arguments)}function r(e){vr=e}function
o(e){return e instanceof Array||"[object Array]"===Object.prototype.toString.call(e)}function a(e){return null!=e&&"[object Object]"===Object.prototype.toString.call(e)}function i(e){var
t;for(t in e)return!1;return!0}function s(e){return"number"==typeof e||"[object Number]"===Object.prototype.toString.call(e)}function u(e){return e instanceof Date||"[object
Date]"===Object.prototype.toString.call(e)}function l(e,t){var n,r=[];for(n=0;n<e.length;++n)r.push(t(e[n],n));return r}function c(e,t){return
Object.prototype.hasOwnProperty.call(e,t)}function d(e,t){for(var n in t)c(t,n)&&(e[n]=t[n]);return c(t,"toString")&&(e.toString=t.toString),c(t,"valueOf")&&
(e.valueOf=t.valueOf),e}function f(e,t,n,r){return gt(e,t,n,r,!0).utc()}function p(){return{empty:!1,unusedTokens:[],unusedInput:
[],overflow:-2,charsLeftOver:0,nullInput:!1,invalidMonth:null,invalidFormat:!1,userInvalidated:!1,iso:!1,parsedDateParts:[],meridiem:null}}function h(e){return null==e._pf&&
(e._pf=p()),e._pf}function m(e){if(null==e._isValid){var t=h(e),n=gr.call(t.parsedDateParts,function(e){return
null!=e}),r=!isNaN(e._d.getTime())&&t.overflow<0&&!t.empty&&!t.invalidMonth&&!t.invalidWeekday&&!t.nullInput&&!t.invalidFormat&&!t.userInvalidated&&
(!t.meridiem||t.meridiem&&n);if(e._strict&&(r=r&&0===t.charsLeftOver&&0===t.unusedTokens.length&&void 0===t.bigHour),null!=Object.isFrozen&&Object.isFrozen(e))return
r;e._isValid=r}return e._isValid}function y(e){var t=f(NaN);return null!=e?d(h(t),e):h(t).userInvalidated=!0,t}function v(e){return void 0===e}function _(e,t){var
n,r,o;if(v(t._isAMomentObject)||(e._isAMomentObject=t._isAMomentObject),v(t._i)||(e._i=t._i),v(t._f)||(e._f=t._f),v(t._l)||(e._l=t._l),v(t._strict)||(e._strict=t._strict),v(t._tzm)||
(e._tzm=t._tzm),v(t._isUTC)||(e._isUTC=t._isUTC),v(t._offset)||(e._offset=t._offset),v(t._pf)||(e._pf=h(t)),v(t._locale)||(e._locale=t._locale),br.length>0)for(n in
br)r=br[n],o=t[r],v(o)||(e[r]=o);return e}function g(e){_(this,e),this._d=new Date(null!=e._d?e._d.getTime():NaN),this.isValid()||(this._d=new Date(NaN)),Mr===!1&&
(Mr=!0,t.updateOffset(this),Mr=!1)}function b(e){return e instanceof g||null!=e&&null!=e._isAMomentObject}function M(e){return e<0?Math.ceil(e)||0:Math.floor(e)}function k(e){var
t=+e,n=0;return 0!==t&&isFinite(t)&&(n=M(t)),n}function x(e,t,n){var r,o=Math.min(e.length,t.length),a=Math.abs(e.length-t.length),i=0;for(r=0;r<o;r++)
(n&&e[r]!=+t[r]||!n&&k(e[r])!=+k(t[r]))&&i++;return i+a}function T(e){t.suppressDeprecationWarnings===!1&&"undefined"!=typeof console&&console.warn&&console.warn("Deprecation warning:
"+e)}function w(e,n){var r=!0;return d(function(){if(null!=t.deprecationHandler&&t.deprecationHandler(null,e),r){for(var o,a=[],i=0;i<arguments.length;i++){if(o="","object"==typeof
arguments[i]){o+="\n["+i+"] ";for(var s in arguments[0])o+=s+": "+arguments[0][s]+", ";o=o.slice(0,-2)}else o=arguments[i];a.push(o)}T(e+"\nArguments:
"+Array.prototype.slice.call(a).join(""))+"\n"+(new Error).stack),r=!1}return n.apply(this,arguments)},n)}function S(e,n){null!=t.deprecationHandler&&t.deprecationHandler(e,n),kr[e]||
(T(n),kr[e]=!0)}function C(e){return e instanceof Function||"[object Function]"===Object.prototype.toString.call(e)}function L(e){var t,n;for(n in e)t=e[n],C(t)?
this[n]=t:this["_"+n]=t;this._config=e,this._ordinalParseLenient=new RegExp(this._ordinalParse.source+"|"+/\d{1,2}/.source)}function D(e,t){var n,r=d({},e);for(n in t)c(t,n)&&
(a(e[n])&&a(t[n])?(r[n]={},d(r[n],e[n]),d(r[n],t[n])):null!=t[n]?r[n]=t[n]:delete r[n]);for(n in e)c(e,n)&&!c(t,n)&&a(e[n])&&(r[n]=d({},r[n]));return r}function E(e)
{null!=e&&this.set(e)}function P(e,t,n){var r=this._calendar[e]||this._calendar.sameElse;return C(r)?r.call(t,n):r}function O(e){var
t=this._longDateFormat[e],n=this._longDateFormat[e.toUpperCase()];return t||!n?t:(this._longDateFormat[e]=n.replace(/MMMM|MM|DD|dddd/g,function(e){return
e.slice(1)}),this._longDateFormat[e])}function Y(){return this._invalidDate}function A(e){return this._ordinal.replace("%d",e)}function j(e,t,n,r){var o=this._relativeTime[n];return
C(o)?o(e,t,n,r):o.replace(/%d/i,e)}function R(e,t){var n=this._relativeTime[e>0?"future":"past"];return C(n)?n(t):n.replace(/%s/i,t)}function I(e,t){var
n=e.toLowerCase();Or[n]=Or[n+"s"]=Or[t]=e}function F(e){return"string"==typeof e?Or[e]||Or[e.toLowerCase()]:void 0}function H(e){var t,n,r={};for(n in e)c(e,n)&&(t=F(n),t&&
(r[t]=e[n]));return r}function N(e,t){Yr[e]=t}function W(e){var t=[];for(var n in e)t.push({unit:n,priority:Yr[n]});return t.sort(function(e,t){return e.priority-t.priority}),t}function
B(e,n){return function(r){return null!=r?(U(this,e,r),t.updateOffset(this,n),this):z(this,e)}}function z(e,t){return e.isValid()?e._d["get"+(e._isUTC?"UTC":"")+t]():NaN}function U(e,t,n)
{e.isValid()&&e._d["set"+(e._isUTC?"UTC":"")+t](n)}function V(e){return e=F(e),C(this[e])?this[e]():this}function q(e,t){if("object"==typeof e){e=H(e);for(var
n=W(e),r=0;r<n.length;r++)this[n[r].unit](e[n[r].unit])}else if(e=F(e),C(this[e]))return this[e](t);return this}function K(e,t,n){var r=""+Math.abs(e),o=t-r.length,a=e>=0;return(a?
n?"+":"":"-")+Math.pow(10,Math.max(0,o)).toString().substr(1)+r}function G(e,t,n,r){var o=r;"string"==typeof r&&(o=function(){return this[r]()}),e&&(Ir[e]=o),t&&(Ir[t[0]]=function()
{return K(o.apply(this,arguments),t[1],t[2])}),n&&(Ir[n]=function(){return this.localeData().ordinal(o.apply(this,arguments),e)})}function J(e){return e.match(/\[[\s\S]/)?e.replace(/^\
[|\]$/g,""):e.replace(/\\/g,"")}function X(e){var t,n,r=e.match(Ar);for(t=0,n=r.length;t<n;t++)Ir[r[t]]?r[t]=Ir[r[t]]:r[t]=J(r[t]);return function(t){var
o,a="";for(o=0;o<n;o++)a+=r[o]instanceof Function?r[o].call(t,e):r[o];return a}}function $(e,t){return e.isValid()?(t=Z(t,e.localeData()),Rr[t]=Rr[t]||X(t),Rr[t]
```

Joonis 3 Koodifragment kliendi seadmes töötavast lahendusest

4.4 Arhitektuur

Esmapilgul läheb tarkvaraarhitektuuri koostamine vastuollu kõigi oluliste agiilse arendamise põhimõtetega [30]. See ei loo konkreetselt ühtegi arendustsüklisse juurde väärtust ning on tavaliselt seotud märksõnadega, nagu pikaajaline planeerimine ja mahukas dokumentatsioon [30]. Ka meeskonnad, kes töötavad pidevalt lahenduse arendamisega ja ilmselt oleks huvitatud võimalikult lihtsast ja efektiivsest arhitektuurist, ei ole võimelised seda saavutama, sest nende põhiline töö keskendub ülesannetele, mis on süsteemi vaatest palju kitsamad [30]. Seega jääb põhiküsimuseks, kuidas saavutada arenduskindlusele vajalik arhitektuuriline püsivus, samal ajal mitte ohverdades agiilseid metoodikaid ja töövõtteid.

Agiilsete metoodikate rakendamise üks olulisemaid eesmärke on muuta arendatav tarkvara vastuvõtlikuks muutustele [26]. Tuleb aga ära tunda, kui selline vastuvõtlikkus pole otstarbekas ega võimalik. Antud töö käigus loodava teenuse pakkumisel peavad teatud komponendid jääma kliendi valdusesse, mis muudab igasuguse arhitektuurilise muutuse komponentide omavahelises suhtluses väga raskesti elluviidavaks. Kuid muutust, mida pole praktiliselt võimalik realiseerida, pole vaja painutada järgima agiilseid praktikaid. Selle tõdemuse tulemusena loodi järgnev komponentide arhitektuur (vt joonis 4), mis peab jääma paika ka pikemas plaanis.



Joonis 4 Süsteemi arhitektuur

Komponentide arhitektuuri mudelis on sees 4 eraldatavat tarkvara. Kontrollseadmed, mis juhivad andurite tööd. Keskseade, mis asub kliendi sisevõrgus, salvestab kontrollseadmete kogutud info ning pakub sisevõrgu piires veebirakendust süsteemi juhtimiseks. Server, mis tegutseb väljaspool sisevõrku ja töötab sillana keskseadme ning muude sisevõrgust väljas asuvate lahenduste vahel, pakkudes lisaks süsteemihaldurile võimalust määrata, millistel keskseadmetel on lubatud ühendada serverisse. Mobiilirakendus, mis võimaldab kontrollida lahendust sõltumata asukohast võrgus, kasutades selleks ära avalikus võrgus asuva serveri funktsionaalsust.

5 Tehnoloogiad

Üks olulisi agiilsuse põhimõtteid on, et inimesed ja suhtlus on tähtsamad kui protsessid ja arendusvahendid [26]. Samas on vaja töö alustamiseks mingisugune valik teha. Seejuures on mõistlik kasutada valiku langetamisel infot hetkeolukorra kohta ning püüda leida mitte ainult võimalikke tehnoloogiaid, vaid optimaalseimaid võimalikke tehnoloogiaid, mis töötaks lahenduse huvides laiemalt kui ainult realiseerimisvahendina.

5.1 Keeled

Programmeerimiskeelte valiku puhul on tähtis vältida valiku paradoksi, mille käigus väikese võidu saamiseks kulutatakse palju aega [31]. Selle eesmärgi saavutamiseks sõnastati konkreetsed nõuded, mida keel peab täitma, et pääseda arutellu ning kindlad kriteeriumid, mille järgi allesjäänud valikuvariante ritta seada.

Nõuded kasutavale keelele olid järgmised:

- Töö autor kui ainus lahenduse elluviija peab olema valitud keelega varasemalt tuttav
- Platvorm peab olema võimeline valitud keelt kasutama

Kriteeriumid, mille alusel järelejäänud valikuid järjestati olid:

- Toetatud platvormide hulk
- Populaarsus
- Hind tööjõuturul

Nõuete rakendamise tulemuseks oli, et kindlasti tuleb kasutada C++, mis on ainuke võimalik valik Arduino platvormile. Ning teistele platvormidele jäid edasisse valikusse Javascript, Java, Python, C++ ja PHP.

Kolmest kasutatud kriteeriumist olid nii populaarsus kui ka hind tööjõuturul kõigi valikusse jäänud keelte puhul suhteliselt sarnased. Populaarsuse alusel mahuvad kõik keeled maailma kümne suurima hulka [32]. Toetudes arendajate seas läbiviidud veebiküsitlusele, ei ole suured ka nende keelte vahelised palgaerinevused Euroopa tööjõuturul [33]. Seega jäi peamiseks kaalukeeleks toetavate platvormide arv. Selles kategoorias aga jäi teistele valikutele selgelt peale Javascript, sest lisaks veebilehitsejas

ainukeseks valikuks olemisele, on Javascripti võimalik kasutada ka platvormiüleseks arenduseks mobiilirakendustes. Nii kasutati lahenduse loomisel, kõikjal peale Arduino platvormi, Javascriptil põhinevaid tööriistu.

5.2 Raamistikud

Raamistikud on vahendid, mis lasevad arendajal keskenduda peamiselt oma äriloogika rakendamisele ja lahendavad ära enamiku arenduste ühisosasse kuuluvad probleemid. Antud projekt kasutab kahte põhilist raamistikku. Esimene on serveri raamistik, mis peaks töötama nii kekseadmes kui ka kekseadmeid muu internetiga ühendavas serveris, et hallata päringuid, nende turvalisust ning andmebaasiühendusi. Teise raamistiku ülesandeks on muuta töö nii mobiilirakenduses kui ka veebilehitsejas ühetaoliseks ning lihtsustada kasutajale interaktiivse info kuvamist.

Serveri raamistikest langes valik Sails.js'ile. Kuigi tegu on raamistikuga, mis pakub võimalusi kogu MVC mustri täielikuks rakendamiseks, on võimalik seda kasutada ka ainult serveripoolse lahendusena. Sails.js'i kasuks otsustamise peamisteks põhjusteks olid oma ORMi toetamine, mis võimaldab tunduvalt lihtsustada andmebaasi päringute tegemist, serveri ja rakenduse kahepoolse suhtluse võimaldamine ning lihtne ligipääsude piiramise süsteem, mis võimaldab väikse vaevaga rakendada äriloogikast tulenevaid nõudeid. Lisaks kuulub Sails.js'i eeliste hulka võimalus sobituda mitme erineva andmebaasisüsteemidega, mis muudab arhitektuuri paremini kohanduvaks ja seega agiilsemaks.

Nagu eelnevalt mainitud, toimus raamistiku valimine veebi- ja mobiilirakendusse ühe otsusena. Ühe või vähemalt väga sarnaste raamistike kasuks otsustamine oli eesmärk, sest vastasel juhul oleks süsteemi tekkinud juurde veel kaks erinevat töökeskkonda ning arendamisprotsessi oleks lisandunud ebavajalikku keerukust. Ühe keskkonna puhul on võimalik vähemalt äriloogika puhul piirduda ühekordse realiseerimisega. Raamistiku valikul mobiili ja veebi keskkonda oli kaks selget võimalikku valiku varianti: Angular2 ja React. Mõlemal neil platvormidel on ka vastavad mobiiliarenduse vasted nimedega Ionic2 ja React Native.

Otsuse langetamisel saigi määravaks teguriks, kuidas antud raamistikud toetavad mobiilirakenduse arendamist. Põhimõtteline erinevus React Native ja Ionic2 vahel on, et esimesega töötades on lõpptulemuseks platvormispetsiifiline lahendus, mis on seetõttu ka

küllaltki kiire, teine aga kasutab nn veebivaadet, mis sisuliselt tähendab rakenduse sees veebilehe kuvamist ja jääb kiiruses React Native'le alla. React Native'i platvormi spetsiifilisust võib näha nii hea kui ka halva küljena, ühest küljest loob see igal platvormil just sellele platvormile omase kasutajaliidese, samas eksisteerib tugi ainult kindlatele komponendile ja kõik ülejäänud tuleb juurde luua kirjutades platvormi spetsiifilist koodi. Kõike eelnevat arvestades sündis otsus kasutada arendamisel React Native raamistikku, kuna vajadus rakendusspetsiifiliste komponentide järgi oli väike ning kiirus osutus kaalukamaks argumendiks.

5.3 Teegid

Teeke kasutati antud lahenduse loomisel peamiselt kahel eesmärgil: säästa aega keerulise funktsionaalsuse realiseerimises ning platvormide lõikes võimalikult sarnase kasutuskogemuse loomiseks. Olulisematena saab välja tuua Material UI, Chart.js'i ning React Camera.

Material UI on algselt React'ile arendatud kasutajaliidese komponentide ja funktsonaalsuste kogum, mis rakendab Google poolt välja töötatud Material Designi põhimõtteid [34]. Vähendades tunduvalt kasutajaliidese disainiks kuluvat aega, laskis Material UI keskenduda rohkem äriloogikale. Lisaks omab vastav teek ka ülekannet React Native'sse, nii on võimalik rakendada sisuliselt samu kasutajaliidese elemente kõikides vaadetes ning uuesti kasutada ühel platvormil loodud lahendusi teistes.

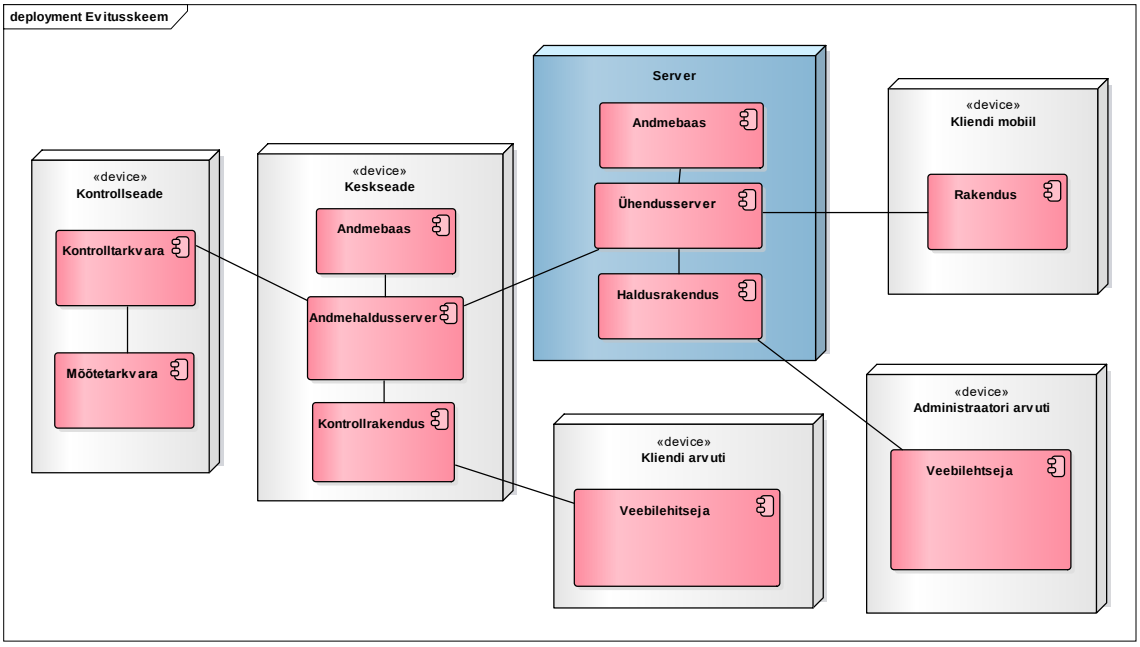
Chart.js [35] on võrgurakenduste kasutajaliidestes kasutatav diagrammide loomist lihtsustav teek. Teegi suurimaks eeliseks oli, et pakutakse dünaamilist andmete muutmist, mis on lahendatud animatsioonidena. See funktsonaalsus reaalajas andmeid kuvaval lahendusel on üliolulline. Ka võimaldab Chart.js muuta diagrammide telgede konfiguratsiooni, et pakkuda kasutajale võimalikult arusaadavat pilti.

React Camera [36] on teek, mis React Native's rakenduse loomisel võimaldab kasutada kaamera funktsonaalsust. Tegu on platvormiülese teegiga, mis tähendab, et korraga on võimalik luua töötav lahendus nii Androidi kui ka iOSi platvormidele. Lisaks elementaarsele pildistamisfunktsionaalsusele sisaldab teek endas ka võimalust skaneerida QR-koodi [36], mis oli antud töö kontekstis mobiilirakenduses oluline funktsionaalsus.

6 Tarkvara

Töö käigus loodud tarkvarast ülevaate andmiseks on kasutatud kahte erinevat vaadet. Komponendivaade, mis annab täpsema ülevaate komponendi tööpõhimõtetest, ja süsteemivaade, mis näitab laiemalt ära süsteemi kui terviku funktsionaalsuse.

6.1 Tarkvarasüsteemi komponendid



Joonis 5 Tarkvara evitusskeem

Koduautomaatika teenuse tarkvara jaguneb laiali neljaks osaks: kontrollseade, kekseade, server ning mobiilirakendus. Jagunemine tuleneb seadmetest, kus tarkvara kasutatakse (vt joonis 5). Seadmepõhiselt on laiali jagatud nii arendamis- kui ka rakendamisprotsess. Eraldamine on vajalik, sest tulenevalt suurest mahust tuleb süsteemi osi näha ja hallata ka iseseisvate tarkvaraprojektidena. Samuti aitab see selgemalt eraldada funktsionaalsusi, mille eest süsteemi erinevad komponendid vastutavad.

6.1.1 Kontrollseade

Kontrollseadmes töötaval tarkvaral on kaks eesmärki: edastada kekseadmele infot riistvara oleku kohta ning realiseerida riistvaras kekseadmest tulevad käsud.

Kontrollseadme lahendust eristab teistest süsteemi komponentidest suur hulk võimalike riistvaralisi variatsioone. See tähendab, et kaks kontrollseadet võivad juhtida või hallata täiesti erinevat riistvara. Arenduse ja rakendamise seisukohast lisab see keerukust, sest tarkvara peab olema võimeline arvestama paljude erinevate stsenaariumitega. Raskuse ületamiseks kasutati igal pool sama tarkvara, kuid kaasa on lisatud konfiguratsioonifail (vt joonis 6), mis sisaldab endas konkreetse seadme riistvara kirjeldust.

```
var config = {};

config.device = {};
config.relay = {};
config.weather = {};
config.greenhouse = {};

config.device.uniqueIdentifier = 'hello world';

config.relay.numberOfChannels = 8;
config.relay.channelsPins = [40, 38, 36, 37, 35, 33, 31, 29];

config.greenhouse.defaultWateringPoint = 200;
config.greenhouse.wateringRelayPin = 7;

module.exports = config;
```

Joonis 6 Kontrollseadme konfiguratsioonifail

Kontrollseadme põhiliseks funktsionaalsuseks võib lugeda järgmiseid keskseadet teenindavaid funktsioone: keskseadmesse ühendamine, ilmainfo ja kastepiiri saatmine, kastepiiri uuendamine, releekanalite väärtuste saatmine ning viimase erijuht releekanali ümberlülitamine.

Kogu nimetatud funktsionaalsuse realiseerimiseks koosneb tarkvara veel eraldi kahest erinevast komponendist: Arduinos rakendatavast C++ koodist analoogsignaalide lugemiseks ning põhifunktsonaalsuse eest vastutavast Node.js koodist Orange Pi Zeros. Lisaks on kasutusel MySql andmebaas, mis tagab, et seadme taaskäivitumisel ei läheks kaduma tööks oluline informatsioon.

6.1.2 Keskseade

Keskseadme tarkvara puhul saab eristada kaht põhilist eesmärki: võimaldada kontrollseadmete töö juhtimist ja pakkuda seda võimekust väljapoole sisevõrku.

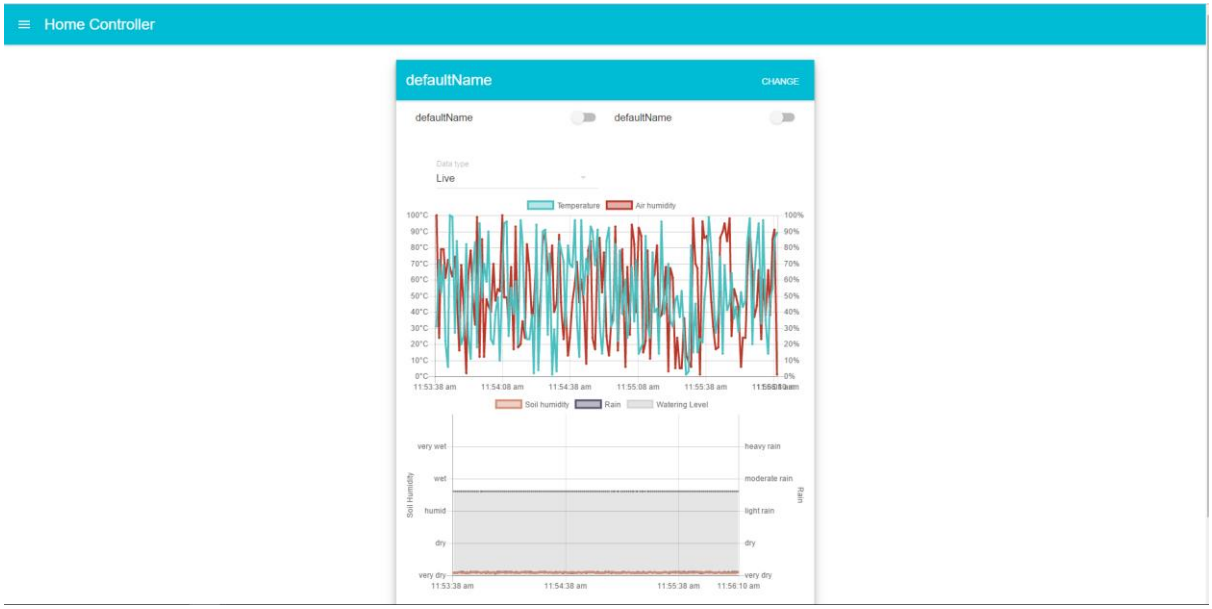
Funktsionaalsus, mida keskseade pakub kontrollseadmete juhtimiseks, kattub suures osas kontrollseadme enda funktsionaalsusega. Sisuliseks vaheks on, et kui kontrollseadmes tegutseti otse riistvaraga, siis keskseade saab infot ja jagab käske vastavalt kasutaja sisenditele. Lisaks kuulub keskseadme alla ka funktsionaalsus, mille eesmärgiks on

kontrollseadmete mugavam haldamine kasutajale. Nendeks funktsioonideks on: kontrollseadmete nimede vaatamine ja muutmine, releekanalite nimede vaatamine ja muutmine, kontrollseadmete kasutava Wi-Fi seadete vaatamine ja muutmine.

Lahenduse realiseerimiseks koosneb keskseadme tarkvara omakorda kolmest alamosast: serveripoolsest lahendusest, eessüsteemist ja ühendusrakendusest internetti.

Sails.js'is realiseeritud serveripoolne lahendus töötleb ja vahendab kontrollseadmete infot eessüsteemi ning edastab eessüsteemist tulevad käsud kontrollseadmetele. Kontrollseadmete info töötlemine tähendab enamikul juhtudel ka salvestamist. Selleks kasutab Sail.js ORMi, mis on konfigureeritud kasutama PostgreSql andmebaasi. Antud andmebaas on ühtlasi ka põhiliseks äriloogikas olulise informatsiooni hoiukohaks. Iga kliendi isikliku keskseadmes asuva andmebaasi eeliseks on, et info ja selle hoidmisega kaasnevad riskid on hajutatud.

React eessüsteem koosneb kahest vaatest: seadmed ja sätted. Seadmete vaates (vt joonis 7) on kujutatud kõiki hetkel aktiivseid keskseadmega ühenduses olevaid kontrollseadmeid, seal on võimalik neid juhtida ja näha kogutud andmeid. Sätete vaates on väljad Wi-Fi seadete muutmiseks ning teksti ja QR-koodi kujul ligipääsukood, mida kasutades saab siduda mobiilirakenduse keskseadmega.



Joonis 7 Keskseadme kasutajaliidese seadmete vaade

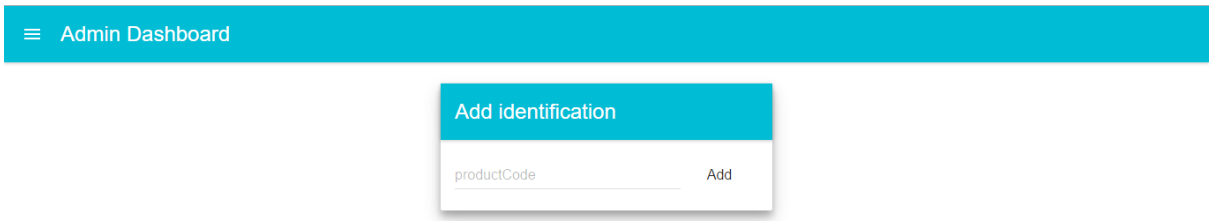
Eraldi saab keskseadmes välja tuua ühendusrakenduse internetis asuva serveriga. Selle loomiseks on kasutatud on kahte sails.socket.js ja Socket.IO tehnoloogiatel põhinevat ühendust. Üks ühendus on loodud avalikus võrgus asuva serveriga ja teine ühendus

keskseadmes asuva serveriga. Rakenduse ülesandeks on kuulata internetist tulenevaid päringuid, edastada need sobival kujul keskseadmesse ning lõpuks saata tulemus tagasi internetis asuvasse serverisse. Tegu on kui virtuaalse eessüsteemiga keskseadme serveri küljes, mis edastab päringuid ja saadab kogu tagasituleva info avalikus võrgus asuvasse serverisse.

6.1.3 Server

Avalikus võrgus asuval serveril on kaks põhieesmärki - võimaldada ligipääsu keskseadme funktsionaalsusele väljaspool kliendi lokaalset võrku ning võimaldada administraatoril kontrollida, millistel keskseadmetel on ligipääsuõigus internetis asuvasse serverisse. Nende eesmärkide täitmiseks on rakendatud järgmine funktsionaalsus: keskseadmetele tootekoodi alusel ühendumisõiguste andmine, õige tootekoodiga keskseadmete ühendumise võimaldamine, ligipääsukoodi alusel kliendirakenduse ja keskseadme vahelise loogilise sideme loomine, administraatori autentimine. Serveri poolt pakutava ühendusfunktsionaalsuse kasutajaks on praeguses süsteemis mobiilirakendus.

Lisaks eelnevalt mainitud serveripoolsele lahendusele on süsteemi haldamise lihtsustamiseks loodud väike eessüsteem, kus administraator saab pärast sisselogimist võrgurakendust kasutades lisada ühendumisõigustega keskseadmeid. See koosneb sisselogimisvaatest ning administraatorivaatest (vt joonis 8), kus praegu ainsaks funktsionaalsuseks on ühendumisõiguste andmiseks tootekoodi sisestamine.

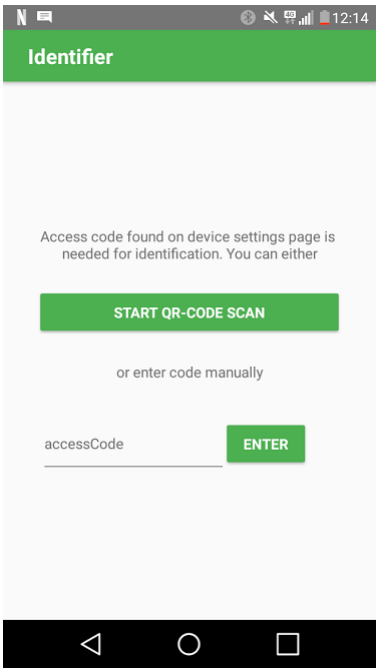


Joonis 8 Serveri administraatorivaade

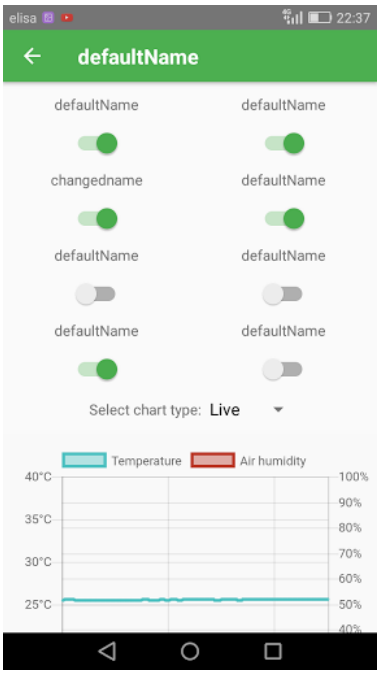
6.1.4 Mobiilirakendus

Mobiilirakendust võib põhimõtteliselt vaadelda kui keskseadme eessüsteemi realisatsiooni mobiili platvormil. Suurimaks erinevuseks on, et kasutamisvõimalus ei piirne kohaliku võrguga ning sellest tuleneb protseduur, kus enne kasutamise alustamist tuleb mobiilirakendus siduda keskseadmega.

Sidumisfunktsionaalsus ongi põhiliseks funktsionaalseks erinevuseks keskseadme eessüsteemi ja mobiilirakenduse vahel ning on inspireeritud töö autori kasutuskogemusest Starmani pakutava televisiooni vaatamiseks mõeldud mobiilirakendusega. Sidumiseks on vajalik, et keskseade on suutnud saavutada ühenduse avalikus võrgus asuva serveriga. Seejärel saab mobiilirakenduse kasutaja, kas QR-koodi kujul sisse skaneerida või käsitsi sisestada (vt joonis 9) keskseadme seadete vaates leiduva ühenduskoodi.



Joonis 9 Sidumisvaade



Joonis 10 Kontrollseadme vaade

Peale sidumisvaate omab mobiilirakendus võrreldes keskseadme eessüsteemiga veel üht lisavaadet. Kui keskseadmes on kõik ühendatud kontrollseadmed ja nende juhtimine ning jälgimine koondatud ühte vaatesse, siis mobiilses seadmes on tulenevalt ekraani mõõtmest eraldi vaadetena kontrollseadmete nimekiri ning iga üksiku kontrollseadme juhtimine ja jälgimine (vt joonis 10).

Nagu eelnevalt mainitud kasutati mobiilirakenduse arendamise tehnoloogiana React Native't, et maksimaalselt kasutada tehnoloogia poolt pakutavat, seetõttu on kõik rakenduse komponendid arendatud nii, et need töötaks nii iOS-il kui ka Androidi platvormidel. Kahjuks tulenevalt töö autori piiratud ligipääsust Apple tehnoloogiale on antud töö käigus valminud lahendusena kättesaadav ainult Androidi versioon.

6.2 Rollid

Lähtudes varasemalt toodud teenusekirjeldusest peab teenus lisaks kliendile kasutuviisidele arvestama ka teenuse osutajaga. See tähendab, et ka tarkvaras peab kajastuma kaks kasutajarolli:

- Klient
- Administraator

6.3 Funktsionaalsus

Komponentide rohkuse ja süsteemi keerukuse tõttu on tarkvaralahenduses loodud palju seadmete omavahelist ning kasutajakogemust toetavat funktsionaalsust. Selle kõrval eksisteerib funktsionaalsus, mis on defineeritud teenusekirjelduses ning võib vaadelda kui terviksüsteemi põhifunktsionaalsust. Selleks on:

Kliendil

- Releede olekute vaatamine
- Releede olekute muutmine
- Ilmainfo vaatamine
- Kasvuhooneinfo vaatamine
- Kasvuhoone kastmispiiri muutmine

Administraatoril

- Keskseadme ligipääsuõiguse andmine

6.4 Paigaldus

6.4.1 Tarkvara

Loodud tarkvara komponentides vajavad pidevalt paigaldamist uutele seadmetele kaks: keskseadme ja kontrollseadme tarkvara. Mõlemad lahendused töötavad Orange Pi Zero seadmetel, mis kasutavad kõvakettana microSD-kaarte. Et vältida olukorda, kus iga seadmega peaks läbi tegema suhteliselt keeruka ja väga sarnase konfigureerimisprotsessi, on loodud täielik koopia kahest microSD-kaardist, kus selline protsess on juba läbi viidud. Nii piisab kontrollseadme puhul, kui kirjutada antud koopia tühjale kaardile ning sisestada seadmesse ning lisada riistvaraga kokku sobiv konfiguratsiooni fail.

Keskseadmel on vajalikuks lisasammuks enne kliendile üleandmist see ühe korra käivitada. Nii genereerib süsteem endale tootekoodi, mida administraator saab kasutada, et anda keskseadmele ligipääsuõigused välisvõrgus asuvasse serverisse.

6.4.2 Seadmed

Seadmete esmakordsel paigaldamisel tuleb alustada keskseadmest. Heaks praktikaks on ühendada see võrku kasutades kaablit, on võimalik kasutada ka Wi-Fit, kuid kuna see pole ettenähtud kasutusjuht, siis on vajalikud tehnilised oskused. Samuti on töö autor sellise ühenduse puhul täheldanud süsteemi töös ebastabiilsust ja ootamatuid hangumisi. Keskseadme esimesel käivitumisel tuleb seadistada kasutajaliideses sisevõrgu Wi-Fi andmed, et võimaldada kontrollseadmetel selle info pärimine. Samuti võib kaaluda ruuteris keskseadmele staatilise IP andmist, nii saab olla absoluutselt kindel, et keskseadme aadress võrgus on püsiv.

Kontrollseadme esmakordsel kasutamisel on samuti vajalik esimene käivitus teha kaablit kasutades. Selle käivituse ajal saab kontrollseade pärida keskseadmelt Wi-Fi info ning ennast vastavalt seadistada. Nii on välditud olukord, kus klient peab iga kontrollseadme seadistamiseks eraldi aega kulutama. Järgnevatel käivitustel suudavad kontrollseadmed ühenduse luua automaatselt Wi-Fi võrku. Siiski tuleb tähele panna, et kui peaks toimuma Wi-Fi seadetes muudatus, mida pole eelnevalt konfigureeritud keskseadme kasutajaliideses, siis on vajalik kõik kontrollseadmed uuesti käivitada kasutades ühenduseks jällegi kaablit.

6.5 Turvalisus

Süsteemi turvalisuses on selgelt eristatud kaks keskkonda: kliendi sisevõrk ja välisvõrk. Sisevõrgus langeb peamine vastutus turvalisuse eest kliendile ehk süsteem on sama ligipääsetav kui kliendi enda sisevõrk. Välisvõrgus liikuva info turvalisuse eest vastutab teenuseosutaja, kes tagab, et kõik ühendused kasutaks HTTPS protokolli ning ühendusi vahendav server oleks töökorras ning mitte kompromiteeritud.

Sellise rõhuasetuse määras peamiselt kaks faktorit. Esiteks on kliendi sisevõrgus turvalise lahenduse loomine ilma oluliselt süsteemi keerukust tõstmata raske ning selle eest vastutust kanda ei ole mõistlik. Teiseks peab teenuseosutaja oma välisvõrgus asuva serveri turvalisuse tagama niikuinii ning HTTPSi kasutamine ei lisa praktiliselt mingit

keerukust. Lisaks võtab teenuseostaja andmete liigutamisega oma süsteemides vastutuse, mida eitada pole võimalik ning on mõistlikum kasutada teenuse turunduspunktina.

7 Lahenduse analüüs

Lahenduse analüüsi seisukohalt on kõige olulisemal kohal kaks küsimust. Millisel määral on arenduse käigus realiseerinud teenuse kirjelduses püstitatud nõuded? Millised on järgmised sammud tarkvaraarenduses, et saavutada teenusega veelgi suurem kooskõla?

7.1 Teenuse toetamine

Peatükis tarkvaraarenduse eesmärgid on välja toodud kindlad põhimõtted, mis olid tarkvaraarenduse põhialuseks. Nende põhimõtete maksimaalseks järgmiseks peab teenuses kasutatav tarkvarasüsteem: omama teenuses püstitatud funktsionaalsust, olema riistvaraliselt kergesti laiendatav, lõppkasutajale lihtsalt kasutatav, automatiseeritult uuenev ning kasutuskogemuse alusel arenev. Teisisõnu peab tarkvara toetama tellimuspõhise hallatud teenuse mudelit võimalikult suures mahus.

Teenuse kirjelduses on esitatud vajaliku funktsionaalsusena relee- ja kasvuhoonejuhtimine ning ilmainfo kogumine. Kõrvutades tarkvarasüsteemi poolt pakutava funktsionaalsusega on näha, et need kattuvad üks-ühele. Lisaks selle on järgitud ka nõuet, et funktsionaalsus peab olema kliendile kättesaadav lisaks sisevõrgule ka avalikust võrgust. Nii võib öelda, et tarkvarasüsteemis on lahendatud teenuse poolt etteseatud funktsioonid ning selles punktis on teenus tarkvara poolt täielikult toetatud.

Rõhk riistvaralise laiendamise lihtsusele oli üks läbivaid suundi kogu tarkvaraarendusprotsessis. Peaasjalikult võib näha seda lahendustes, mis on realiseeritud kontrollseadmetel, keskseadmes ja mobiilirakenduses, nende infovahetuses on selgelt eraldatud erinevaid riistvaralisi funktsioone täitvad blokid ning uue riistvara lisamine ei sega teiste blokkide tööd. Parima näite võib leida kontrollseadmetes, kus sõltumata riisvarast töötab kõik sama loogika alusel ning riistvaralist erinevust kontrollitakse vaid eelnevas peatükis kirjeldatud konfiguratsioonifailiga. Tõeliselt saab hinnata, kas tegu oli eduka lahendusega alles pärast riistvaralise funktsionaalsuse kasvu, kuid vähemalt esimese sprindi piires töötas selline rõhuasetus edukalt.

Sarnaste tarkvarade analüüsist töö alguses selgus, et mida väiksem on süsteemi riistvaraline funktsionaalsus, seda lihtsam peab olema lõppkasutaja kogemus. Sellest

printsiibist lähtudes loodi tarkvara nii, et paigaldamisprotsessi käigus poleks vaja tavakasutajal iga seadme ühendamiseks lisavaeva näha. Nii puuduvad lahenduses konfiguratsiooni failid, mida lõppkasutaja peab täitma ning süsteemi tööks vajalikud andmed (nt Wi-Fi seaded) suudavad kontrollseadmed pärida iseseisvalt keskseadmest. Nii võib öelda, et vaadeldes võrdluses esinenud tarkvarasid on antud töös arendatud lahenduses süsteemi paigaldamine tunduvalt lihtsam ja vähesed riistvaralised võimalused edukalt kompenseeritud.

Automatiseeritult uuenemine oli üks eesmärk, millele samuti, eriti tarkvaratehnilises võtmes, pöörati olulist tähelepanu. Antud töö raames loodud süsteemis said selle võimekuse kontrollseadmed ja keskseadmed. Siiski ei ole nende seadmete tarkvaraarendus automatiseeritud täielikult ilma lünkadeta. Samuti ei ole jõutud sarnaste praktikate kasutamiseni teistes süsteemi komponentides. Kuid nendest puudustest hoolimata võib öelda, et minimaalne automatiseeritult uuenemise nõue süsteemi töö stabiilsuse tagamiseks on realiseeritud.

Suuresti jäi tarkvaraarendusel skoobist välja kliendi kasutuskogemuse kogumine ja salvestamine. Süsteemis küll eksisteerib võimekus luua kliendilt tagasiside saamise funktsionaalsus, kuid selle rakendamine esimese sprindi sisse ei mahtunud. Seetõttu võib möönda, et tarkvarasüsteem pole võimeline antud arenduse valminud mahu piires osutama teenusele kliendi tagasiside osas praktilist tuge ning teenuse toetamine on puudulik.

Kasutuskogemuse kogumise, salvestamise ja rakendamise ebaõnnestumise põhjusena saab välja tuua süsteemi arhitektuuri. On langetatud teadlik valik, kus kasutajatel ei paluta avada oma porte, et võimaldada arendajatel süsteemi siseneda välisvõrgust ja sellisel viisil vajalikke andmeid koguda. Tuleb ka märkida, et sellise arhitektuurivaliku lahendamine teisiti oleks olnud võimatu, sest siis oleks rikutud printsiipi, mille alusel peab jääma lõppkasutajale võimalikult vähe konfigureerimisülesandeid. Nagu selgus tarkvaraarenduse käigus, oli tegu põhimõtteliselt vastandlike eesmärkidega, mille korraga lahendamine sellise töömahu juures oli võimatu.

Suuremas plaanis võib siiski öelda, et loodud tarkvara saab hakkama enamiku teenuse poolt püstitatud nõuetega. Kõige olulisemad eesmärgid, nagu realiseeritud funktsionaalsus ja uue riistvara lisamise lihtsus, on täidetud. Tagantjärele võib öelda, et valukohaks osutunud kasutusinfo kogumine jäi lahendamatuks, sest ainuke viis antud nõude lahendamiseks mõistliku aja jooksul oleks olnud lahendada arhitektuur teisiti ning

paluda välisvõrguga suhtluseks avada kliendil ruuteris keskseadmele vajalikud pordid. Nii oleks aga vastuollu mindud jällegi minimaalse konfigureerimise printsiibiga. Seetõttu saab järeldada, et antud tingimustes oli tegu vastandlike nõuetega ning ühe õnnestumine oli vältimatu.

7.2 Tarkvaraarenduse edasised suunad

7.2.1 Testimine

Üks olulisemaid agiilse arenduse aspekte, mis töö autor teadlikult tarkvaraarenduse esimesest faasist välja jättis, oli testimine. Peamiseks ajendiks sellise otsuse puhul oli ebastabiilne koodibaas, mis tulenes süsteemi tarkvaraliste komponentide vahelisest suurest integratsioonide hulgast. Nii võis arenduse käigus juhtuda, et täiesti uute tarkvaraliste osade sissetoomine, muutis oluliselt juba eksisteerivate osade funktsionaalsust. Testimine keskkonnas, kus isegi põhifunktsionaalsus on muutlik, oleks võinud halvemal juhul tähendada, et kehvad tarkvaralised valikud oleks jäänud liiga suure ümbertegemise ettekäändel süsteemi sisse.

Olukord on muutunud pärast esimest sprinti, kus süsteemi komponentide vaheline koostöö on saanud kindlad raamid ning ka kood on muutnud oluliselt stabiilsemaks. Testimine on muutunud praktiliseks ning seda on ka vaja, et mitte lõhkuda töötavat funktsionaalsust uue lisamisega ja tagada võimalikult väike vigade hulk valmivas tarkvaras. See tähendab, et enne edasise funktsionaalsuse loomist tuleb järgnevas tarkvaraarenduses võtta suund olemasoleva funktsionaalsuse testidega katmiseks. Nii jääb tulevikuks ka lahtiseks võimalus liikuda edasi testipõhise arendamisega.

7.2.2 Riistvaralise mitmekesisuse suurendamine

Eelnevast võrdlusest teiste sarnast funktsionaalsust pakkuvate tarkvaradega ei saa märkamata jätta, et viimased toetavad tunduvalt rohkem erinevaid riistvaralisi lahendusi, mida koduautomaatika süsteemiga juhtida. Selle probleemi korvamiseks peab järgneva funktsionaalsuse arendamise keskmes olema eelkõige riistvaralise toe arendamine, jättes hetkeks tagaplaanile kliendipoolsete administratiivsete võimaluste laiendamise. Ühtlasi aitab erinvate kontrollseadmete konfiguratsioonide hulga suurendamine luua kõige kiiremini kliendile olulist lisandväärtust ning laiendada teenuse turgu.

Konkreetselt kuulub riistavaralise toe laiendamise plaani alla lähemas perspektiivis kaks seadet: elektripistiku kontroller ja liikumisandur. Mõlemad seadmed on riistvaraliselt odavad ning juba loodud tarkvarale toetudes kerged realiseerida. Samuti on tegu väga laiu kasutusviise omavate seadmetega, mis täiendavad oluliselt kliendi kasutuskogemust ning sellest tulenevalt täidavad ka lisaväärtuse pakkumise eesmärgi.

7.2.3 Administraatori rolli laiendamine

Tarkvarasüsteemi analüüsist teenuse mudelist lähtuvalt joonistus välja, et hallatud teenusele on võimalik liikuda oluliselt lähemale, kui eksisteeriks võimekus saada ja kasutada kliendi tarkvarakasutusinfot. Nii ongi üheks süsteemi edasiarendussuunaks administraatori rolli suurendamine ning sellele rollile tagasiside haldamiseks ja analüüsiks vajaliku funktsionaalsuse loomine. See annaks teenuseosutajale võimekuse paremini mõista klientide poolt kirjeldatud probleeme ning väärtuslikku informatsiooni vigade kohta, mida saab hiljem kasutada kvaliteetsema tarkvara arendamisel.

Praktilises tähenduses on esimene eesmärk anda süsteemi administraatorkasutajale võimekus tutvuda kliendi keskseadmete ja kontrollseadmete logifailidega ning jälgida reaalajas kliendipõhiselt keskseadme ja avalikus võrgus asuva serveri vahelist suhtlust. Esimene lubaks teenuseosutajal näha täpselt, millistes kohtades, millal ja miks on tekkinud probleemid. Teine laiendus lubaks ära kasutada juba niikuinii läbi avaliku võrgu liikuvat informatsiooni ning olla abiks kliendipoolse süsteemi töö hindamisel.

8 Kokkuvõte

Töö peamiseks eesmärgiks oli tuvastada kirjelduse põhjal teenusepõhise ärimudeli seatavad nõuded koduautomaatika tarkvarale ning hinnata nende alusel koostatud tarkvara kokkusobivust algselt püstitatud eesmärkidega. Selle tulemusena valmis töö käigus neljast erinevast komponendist koosnev koduautomaatika süsteem, mis sisaldab kontrollseadet, keskseadet, välisvõrgus asuvat serverit ning mobiilirakendust. Lisaks tarkvara väljaarendamisele pöörati tähelepanu ka tarkvaraarendusprotsessile ning fikseeriti töövõtted, mis on abiks järjepidevalt kvaliteetse teenuse osutamisel.

Töö tulemusena analüüsiti järgnevate teenusepoolsete nõuete elluviimist tarkvaras - teenuses kirjeldatud funktsionaalsuse omamist, modulaarselt laiendatavust, minimaalse konfigureerimise vajalikkust, automaatset uuenemist ja kasutajakogemuse arvestamist arendamisel. Tarkvaraarendus täitis viiest püstitatud eesmärgist neli: teenuses kirjeldatud funktsionaalsuse omamine, riistvaraliselt modulaarselt laiendatavus, lõppkasutajalt minimaalse seadistamise vajamine ja automatiseeritult uuenemine. Ainsana ei saanud rahuldaval määral kaetud kasutajakogemuse arvestamine arendusprotsessis. Seda eelkõige arhitektuuriliste valikute ja põhimõttelise sobimatuse tõttu teise minimaalse konfigureerimise vajaduse nõudega.

Lõputöö tulem tarkvara ja selle arendusprotsessina leiab praegu kasutust teenuse arenduse edasistes etappides, kus pannakse paika teenuse osutamise infrastruktuur ning disainitakse teenuse komponentide füüsilist kuju. Arenduse eduka lõpetamise korral jõuab teenus turule OÜ Heltico ja OÜ Inseneritööde ühistööna.

Kasutatud kirjandus

[1] „Angular,“ Google, [Võrgumaterjal]. Available: https://angular.io/. [Kasutatud 12 mai 2017].

[2] „API,“ Wikimedia Foundation, [Võrgumaterjal]. Available: https://et.wikipedia.org/wiki/Rakendusliides. [Kasutatud 12 mai 2017].

[3] „Arduino Nano,“ Arduino, [Võrgumaterjal]. Available: https://www.arduino.cc/en/Main/arduinoBoardNano. [Kasutatud 12 mai 2017].

[4] S. Guckenheimer, „What is Continuous Integration?,“ [Võrgumaterjal]. Available: https://www.visualstudio.com/learn/what-is-continuous-integration/. [Kasutatud 12 mai 2017].

[5] „Ionic,“ Wikimedia Foundation, [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Ionic_(mobile_app_framework). [Kasutatud 12 mai 2017].

[6] „Logfile,“ Wikimedia Foundation, [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Logfile. [Kasutatud 12 mai 2017].

[7] „MVC,“ Wikimedia Foundation, [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller. [Kasutatud 12 mai 2017].

[8] „Mysql,“ Wikimedia Foundation, [Võrgumaterjal]. Available: https://et.wikipedia.org/wiki/MySQL. [Kasutatud 12 mai 2017].

[9] „Node.js,“ Wikimedia Foundation, [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Node.js. [Kasutatud 12 mai 2017].

[10] „Orange Pi Zero,“ Xunlong Software CO, [Võrgumaterjal]. Available: http://www.orangepi.org/orangepizero/. [Kasutatud 12 mai 2017].

[11] „ORM,“ Wikimedia Foundation, [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Object-relational_mapping. [Kasutatud 12 mai 2012].

[12] „PostgreSQL,“ Wikimedia Foundation, [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/PostgreSQL. [Kasutatud 12 mai 2017].

[13] „QR-Code,“ Wikimedia Foundation, [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/QR_code. [Kasutatud 12 mai 2017].

[14] „React,“ Facebook, [Võrgumaterjal]. Available: https://facebook.github.io/react/. [Kasutatud 12 mai 2017].

[15] „React Native,“ Dacebook, [Võrgumaterjal]. Available: https://facebook.github.io/react-native/. [Kasutatud 12 mai 2017].

[16] „Raspberry Pi,“ Wikimedia Foundation, [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Raspberry_Pi. [Kasutatud 12 mai 2017].

[17] „Sails.js,“ Sails.js, [Võrgumaterjal]. Available: http://sailsjs.com/. [Kasutatud 12 mai 2017].

[18] „Teek,“ Wikimedia Foundation, [Võrgumaterjal]. Available: https://et.wikipedia.org/wiki/Teek. [Kasutatud 12 mai 2017].

[19] M. A. Rappa, „The utility business model and the future of computing services,“ *IBM systems journal,* kd. 43, nr 1, pp. 32--42, 2004.

[20] D. McMillan, B. Brown, A. Sellen, S. Lindley ja R. Martens, „Pick up and play: understanding tangibility for cloud media,“ %1 *Proceedings of the 14th International Conference on Mobile and Ubiquitous Multimedia*, Linz, 2015.

[21] *Introduction to Managed Services,* CA Technologies, 2012.

[22] openHAB, „OpenHAB introduction,“ openHAB, [Võrgumaterjal]. Available: https://www.openhab.org/introduction.html. [Kasutatud 27 aprill 2017].

[23] pimatic, „pimatic,“ pimatic, [Võrgumaterjal]. Available: https://pimatic.teamemo.com/. [Kasutatud 30 aprill 2017].

[24] „ImperiHome,“ Evertygo SAS, [Võrgumaterjal]. Available: http://www.evertygo.com/imperihome. [Kasutatud 30 aprill 2017].

[25] Home Assistant, „Home Assistant avaleht,“ 30 aprill 2017. [Võrgumaterjal]. Available: https://home-assistant.io/.

[26] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick , R. Martin, S. Mellor, K. Schwaber, J. Sutherland ja D. Thomas, *Manifesto for Agile Development*, 2001.

[27] D. Leffingwell, Agile Software Requirements, Pearson Education, Inc, 2011.

[28] P. Rodríguez, A. Haghighatkhah, L. E. Lwakatare, S. Teppola, T. Suomalainen, J. Eskeli, T. Karvonen, P. Kuvaja, J. M. Verner ja M. Oivo, „Continuous deployment of software intensive products and services: A systematic mapping study,“ *Journal of Systems and Software*, nr 123, pp. 263-291, jaanuar 2017.

[29] H. H. Olsson, H. Alahyari ja J. Bosch, „Climbing the "Stairway to Heaven" -- A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software,“ %1 *38th Euromicro Conference on Software Engineering and Advanced Applications*, 2012.

[30] E. Hadar ja G. M. Silberman, „Agile Architecture Methodology: Long Term Strategy Interleaved with Short Term Tactics,“ %1 *OOPSLA Companion '08*, Nashville, 2008.

[31] B. Schwartz, The Paradox of Choice - Why More Is Less, Harper Perennial, 2004.

[32] S. Cass, „The 2016 Top Programming Languages,“ 26 juuli 2016. [Võrgumaterjal]. Available: http://spectrum.ieee.org/computing/software/the-2016-top-programming-languages.

[33] Stack Overflow, „Stack Overflow Developer Survey Results 2017,“ Stack Overflow, [Võrgumaterjal]. Available: https://stackoverflow.com/insights/survey/2017#top-paying-technologies. [Kasutatud 2 aprill 2017].

[34] Material UI, „Material UI,“ [Võrgumaterjal]. Available: http://www.material-ui.com/#/. [Kasutatud 11 aprill 2017].

[35] Chart.js, „Chart.js documentation,“ Chart.js, [Võrgumaterjal]. Available: http://www.chartjs.org/docs/. [Kasutatud 13 aprill 2017].

[36] lwansbrough, „Github React Native Camera,“ Github, [Võrgumaterjal]. Available: https://github.com/lwansbrough/react-native-camera. [Kasutatud 13 aprill 2017].