TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Valeri Randalainen 142680IAPB

RASPBERRY PI 3 MODEL B WI-FI SEADISTAMISPROTSESSI LIHTSUSTAMINE

Bakalaureusetöö

Juhendaja: Roger Kerse
MSc

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: Valeri Randalainen

22.05.2017

Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks on teha Adobe PhoneGap raamistiku kasutades mobiilirakendust, mis võimaldab Raspberry Pi 3 Model B (edaspidi Raspberry) ühendust nutitelefoniga sätestamaks WiFi ühendust Raspberry peal. WiFi seadistamiseks peab kasutaja looma Raspberry'ga bluetooth ühendust. Peale seda saab rakenduses valida Raspberry't mida hakatakse seadistama, ning ülejäänud protsess WiFi seadistamiseks on sarnane samasuguse protsessiga Android operatsioon süsteemis.

Bakalaureusetöö peamiseks probleemiks on graafilise kasutajaliideseta Raspberry seadistamine. Tavaliselt vajab see mahukaid teadmisi operatsioonisüsteemide administreerimisest. Juhtudel, kui süsteem on krüpteeritud ja/või sinna ei saa sisselogida läbi traadiga võrgu – erinevate seadistuste muutmine saab peaaegu võimatuks. See on probleemiks ka firmadele, kes pakkuvad Raspberry koduautomatisatsiooni juhitava moodulna.

Probleemi lahendamiseks on tehtud kaks rakendust, esimene on mobiilirakendus Adobe PhoneGap raamistikus ja teine rakendus on Raspberry jaoks ning tehtud Node.js tehnoloogiaga. Rakendused suhtlevad omavahel läbi bluetooth serial liidese. Mobiilirakendus on kasutajasõbraliku liidesega ning lubab kiiresti ja mugavalt seadistada WiFi't. Mõlemad rakendused toetavad lisamooduleid, seega saavad arendajad kiiresti lisada süsteemile uut funktsionaalsust.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 30 leheküljel, 5 peatükki, 18 joonist, 0 tabelit.

Abstract

Simplification of Wi-Fi setup process for Raspberry Pi 3 Model B

The purpose of this thesis, is to make two applications, one for mobile platforms and another for Raspberry Pi 3 Model B. Running these two applications on corresponding platforms user should be able, to setup WiFi connection on Raspberry through mobile application after pairing them using Bluetooth. Mobile application should be made using Adobe PhoneGap framework for being able, in perspective, to catch wider user audience.

The main problem this thesis aims to solve is high entrance barrier for using Raspberry with operational system that does not have graphical user interface. Managing such systems requires experience and knowledge in operational system administration field. It also becomes almost impossible to make any changes to operational system from the outside if it is encrypted. However, system managing functionality may be needful for those, who use Raspberry as a home automation central device. Especially in case when such device is provided by company that sells device sets for home automation.

Pointed problem was solved by developing two applications. One is hybrid mobile application that can be built for both Android and iOS. Other is Node.js based application for Raspberry, that can communicate with mentioned mobile application. Communication is bidirectional and allows to transfer any kind of data, as long, as it can be presented in JSON format. Both applications have modular structure allowing any developer to easily connect their modules to application.

As an example, showing this system potential, WiFi setup module was developed for both mobile and Raspberry applications. Module allows to setup WiFi for Raspberry almost in the same uncomplicated way as it can be done on popular mobile platforms.

The thesis is in Estonian and contains 30 pages of text, 5 chapters, 18 figures, 0 tables.

Lühendite ja mõistete sõnastik

AMD	*Advanced Micro Devices,* firma, mis tegeleb arvutiosade tootmisega
API	*Application Program Interface,* rakendusliides
APT	*Advanced Package Tool,* tööristade komplekt, mis on mõeldud Debian operatsioon süsteemide pakettide haldamiseks
CLI	*Command Line Interface*, käsurea liides
CSS	*Cascading Style Sheets,* keel, milles märgitakse üles HTML keelsete failide kujundust
DOM	*Document Object Model,* platvormi ja keele neutraalne liides mis võimaldab programme ja scripte dokumendi sisu manipuleerida
GIT	Versioonihaldustarkvara avatud lähtekoodiga
GPIO	*General-purpose input/output*, universaalne liides andmete edastamiseks/saavutamiseks
GUI	*Graphical user interface*, graafiline kasutajaliides
HTML	*HyperText Markup Language*, veebilehtede märgendamise keel
IDE	*Integrated development environment,* sisseehitatud programmeerimiskeskkond
JSON	*JavaScript Object Notation,* struktuur andmete esitamiseks
MAC adress	*Meda Access Control,* iga seadme unikaale riistvara põhine aadress
NPM	*Node.js package manager,* Node.js pakettide haldamis süsteem
Raspberry	Raspberry Pi 3 Model B, umbes panga-kaardi suurune ühest trükkplaadist kosnev arvuti
SSH	*Secure Shell*, krüptograafiline protokoll turvalise ühenduse loomiseks
URL	*Uniform Resource Locator,* internetiaadress
WiFi	*Wireless Fidelity,* traadita internet

Sisukord

Jooniste loetelu

1 Sissejuhatus

Raspberry on täielik arvuti kaasaegses mõttes, tema eelised ja põhilised omadused on väike suurus 85.60 mm × 56.5 mm × 17 mm ja suhteliselt odav hind $35 [1]. Puudus on aga selles, et ta ei ole niivõrd võimekas nagu süle- või lauaarvuti – selleks, et seda probleemi nivileerida kasutatakse Raspbian Lite [2] - Linux operatsioon süsteemi ilma graafilist kasutajaliidest. Oma omadustega Raspberry sobib hästi kodu automatiseerimissüsteemi juhtseadmeks. Läbi juhtseadme peab toimuma kogu automaatika seadistamine, aga juhtseadme seadistamist võib olla väga raske või võimatu teostada mitmetel põhjustel. Juhtseadme opertastioonisüsteem võib olla krüpteeritud, kasutajal võib vastavatest oskustest puudu olla, võimatu ühendada Raspberry traadiga võrguga. Kõikide mainitud probleemide lahendamine aitab inimestele lihtsamini alustada kas oma kodu automatiseerimisega või lihtsustada kodu automaatika arendamis protsessi arendajatele.

Arvutite kasutamine muidugi ei piiru kodu automatiseerimisega. Kui Raspberry’t ilma graafilist kasutajaliidest oleks lihtsam ja mugavam kasutada siis sobiks see ka tehnoloogia entusiastidele ja lastele keda huvitab automatiseerimine ja/või programmeerimine. Näiteks kui ühendada Raspberry GPIO’ga nuppu, dioodi või kõlari - siis saab sellest teha juba täieliku seadme unikaalsete võimalustega.

Erinevatele probleemidele üldise lahendamise leidmiseks tekkis idee luua kaht rakendust, üks Raspberry ja teine nutitelefoni jaoks. Rakendused suhtlevad omavahel bluetooth serial liidese kaudu ning edastavad teineteisele käske ja andmeid. Mobiilirakenduse kasutajaliides on seeni lihtsustatud, et seadistamise protsess ei oleks keerulisem kui näiteks Android operatsioon süsteemil telefoni seadistamine.

1.1 Eesmärgid

Esimeseks eesmärgiks on luua kahtks rakendust ja seadistada nende omavaheline suhtlemistne. Üks rakendus on mobiilirakendus Adobe PhoneGap [3] raamistikus. Teine

on Raspberry server rakendus Node.js keskkonnas. Rakenduste põhifunktsionaalsuse on alljärgnev:Raspberry server rakendus:

- Võtab vastu nutitelefonist saadetud käsu
- Suunab käsu vastavasse moodulisse, mis seda käsku täidab
- *(Kui käsk nõuab)* Saadab tagasi käsku täitmisel saadud andmeid

Nutitelefoni rakendus:

- Annab valiku ja ühendab valitud Raspberry'ga
- Annab moodulite valiku ja kuvab kasutajale valitud mooduli seadistusi
- Edastab modulist saadud käske Raspberry'le
- *(Kui käsk nõuab)* Ootab, võttab vastu ja edastab vastavale moodulile käsku täitmise tulemusena saadud andmeid

Teiseks eesmärgiks on teha rakendustele WiFi seadistamis mooduli, mis omakorda lubab läbi nutitelefoni rakenduse seadistada WiFi ühendust Raspberry peal, kus jookseb server rakendus.

1.2 Metoodika

Nutitelefoni rakenduse loomiseks otsustasin kasutada Adobe PhoneGap raamistiku, kuna see võimaldab ühekordselt koodi kirjutades genereerida rakendusi erinevatele platvormidele, nendest näiteks üldkasutatavad Android ja iOS. Rakenduse ehitamine on antud juhul sarnane veebilehe kirjutamisega ehk vajab HTMLi ja JavaScript'i teadmist. Raspberry server rakendus on tehtud Node.js keskkonnas ja seega kirjutatud enamasti JavaScript'is.

1.3 Ülesehitus

Käesoleva töö teine peatükk annab ülevaadet arendamisel kasutatud tehnoloogiatest, mõnele tehnoloogiale on lisatud ka vastava tehnoloogia valimise põhjendus. Kolmandas peatükis on kirjeldatud hübriidmobiilirakenduse loomise protsess, mis on omakorda

jagatud rakenduse erinevate osade loomise protsesside kirjelduseks. Neljas peatükk kirjeldab Raspberry serverrakenduse loomist, mis koosneb loomise protsessi kirjeldusest ning tähtsamate arhitektuuriküsimuste lahenduste kirjeldusest.

2 Arendamisel kasutatud tehnoloogiad

Järgnev peatükk kirjeldab tehnoloogiaid, mis otseselt puutuvad rakenduste süsteemi töö- ja loomisprotsessi. Eesmärgiks on tekitada teoreetilist tausta, millega oleks võimalik aru saada rakenduse arhitektuuri ja realiseerimise ideest. Lisaks on kirjeldatud iga teoreetilise osa tähtsus ja/või roll antud töö raames.

2.1 Raspbian Lite

Rasbian on Debian'i põhiline operatsioon süsteem Raspberry Pi jaoks. Raspberry Pi Foundation ametlikult esitab seda nagu põhilist operatsioonisüsteemi kogu Raspberry Pi monoplaatarvutite perekonnale. Esimene versioon sai valmis 2012 aasta juunis. Siiamaani Raspbian operatsioonisüsteemi jätkuvalt aktiivselt arendatakse. Raspbian on hästi optimiseeritud spetsiaalsetele, madala jõudlusega, Raspberry Pi spetsiifilistele, AMD tsentraal protsessoritele. [3]

Raspbian'ist on hetkel kaks erinevat versiooni: RASPBIAN JESSIE WITH PIXEL ja RASPBIAN JESSIE LITE [2]. Esimesel on sisseehitatud graafiline kasutajaliides PIXEL ja teisel on kasutajaliides konsooli põhine. Jessie nime sees tähendab Debian versiooni millel põhineb Raspbian. Antud bakalaureusetöö kirjutamise hetkeks Jessie on kõige uuem avalikult kättesaadav Debian'i versioon. [4]

Oluline on mainda seda, et Raspbian on Debian'i põhine ja seega kasutab pakettide haldamiseks APT süsteemi. Interneti ühenduse olemasolul APT võimaldab ühe käsuga paigaldada tarkvarat mis on antud süsteemis paigadatav [5]. Sellest on palju kasu arendajatele, kuna lisatarkvara kasutamine tihtipeale oluliselt lihtsustab nende tööd.

2.2 Adobe PhoneGap ja Framework7

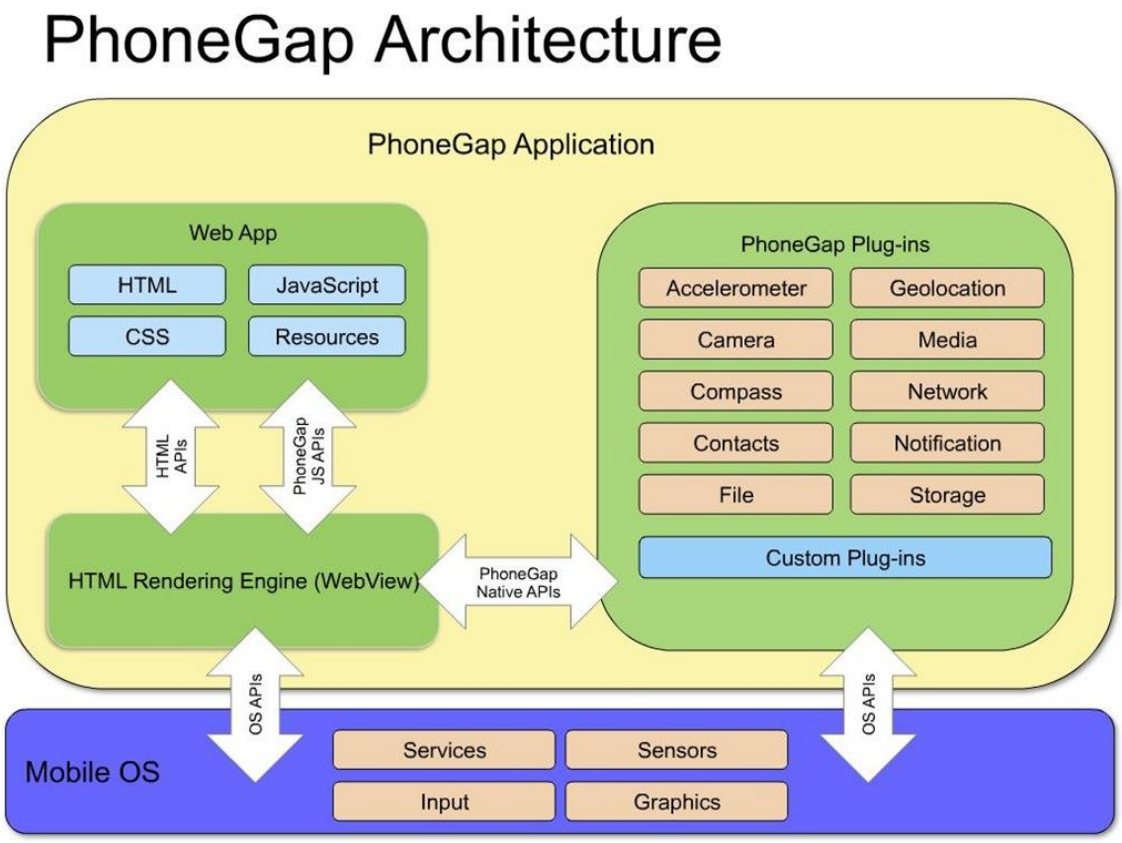
PhoneGap on sisuliselt Apache Cordova'l põhinev raamistik Adobe poolt juurdelistatud funktsionaalsusega [6]. Apache Cordova on üks kõige populaarsematest raamistikutest hübriidrakenduste loomiseks. Hübriidrakenduste idee seisneb selles, et arendaja kirjutab üht rakendust, mis iseenesest on veebileht, ja genereerib rakendusi erinevate platvormide jaoks kasutades Apache Cordova töövahendit.

Framework7 on tasuta ja avaliku lähtekoodiga HTML raamistik hübriidmobiilirakenduste loomiseks. See lihtsustab mobiilirakenduste disani loomist pakkudes juba valmisolevaid komponente, mis oma disaini poolt klappivad kokku nii iOS'i kui ka Android'ile omase disainiga. [7]

Antud töö raames kasutatakse Adobe PhoneGap'i koos Framwork7 raamistikuga. Raamistiku kombinatsioon võimaldab teha kasutajasõbralikku disaini ning luua hea kasutajakogemust keskendudes samal ajal rakenduse sisul.

2.2.1 Hübriidrakenduste arhitektuur Adobe PhoneGap'i näitel

Hübriidmobiilirakenduste loomisel, ei pea programmeerija erinevate platvormide jaoks eraldi koodi kirjutama. Üks ainus HTML, CSS ja JavaScript tehnoloogiatega kirjutatud veebilehet kompilleeritakse PhoneGap'iga erinevate platformide jaoks. [8]



Joonis 1. PhoneGap rakenduse arhitektuur [9]

PhoneGap'i arhitektuuri illustreerib Joonis 1 ja selle täpsustamiseks tuleks mainida, et programmeerija poolt kirjutatud kood asub *Web App* osas. Antud töös arendatav

hübriidrakendus kasutab Bluetooth'i andmete vahetamiseks, mis kuulub vastavalt *Custom Plug-ins*[1] osale.

Võimalus kasutada kolmada isiku poolt tehtud pistikprogramme laiendab programmeerija võimalusi, aga sellel on ka omad puudused. Esiteks, tuleb pöörata tähelepanu sellele, mis on kasutatava koodi litsens ja teiseks, mis platvormide jaoks on see kood kirjutatud. Kogemusest võib öelda, et enamus pistikprogrammidest on kirjutatud veebilehitseja ja Android'i jaoks ning kolmandal kohal on iOS.

2.2.2 Framework7

Framework7 on täiesti nullist[2] kirjutatud raamistik veebi kasutajaliideste arendamiseks. Lisaks kasutamisvalmidele disianielementidele, pakub Framework7 navigatsiooni haldamist ja DOM7 funktsionaalsust. DOM7 on spetsiaalselt antud raamistikule kirjutatud teek, mis effektiivsemal viisil manipuleerub tavalise DOM'iga. [7]

Põhilised objektid, mida antud raamistik pakub ja mis olid kasutatud hübriidrakenduse tegimisel:

- Vaade – omab eraldiseisva navigatsiooni ajalugu ja loogiliselt eraldab rakenduse funktsionaalset osa.
- Leht – osa vaadest, mida näidatakse kasutajale. Korraga saab ühes vaades aktiivne olla ainult üks leht.
- Navigatsioon – vaadete ja lehtede vahelist navigeerimist ja selle navigatsiooni ajalugu haldab täielikult Framework7.
- Modaalne aken – hüpikaken, mis blokeerib kasutaja jaoks rakenduse kasutamist seni, kui kasutaja seda kinni ei pane, või kuni rakendus ise seda ei tee.
- Kaart – informatsiooni näitamiseks mõeldud plokk, mille ülaosas on pilt pealkirjaga ja alaosas asub sellele juurde kuuluv tekst.

---

[1] *Custom Plug-in* – enda või kolmanda isiku poolt tehtud pistikprogramm, mis ei ole vaikimisi toetatud pistikprogrammide hulgas

[2] Framework7 kodulehel [7] on mainitud, et antud raamistik ei kasuta ühtegi teist raamistiku

- Loend, nupp, sisend vormid – tavapärased elemendid iga rakenduse jaoks.

Kõik mainitud objektid on adaptiivsete stiilidega. See tähendab, et nende suurus muutub vastavalt sellele, kui suur on ekraan, kus neid kuvatakse. Tänu adaptsiivsetele stiilidele saab ühekordselt rakendust disainides olla kindel, et enamuse seadmete peal see hakkab välja nägema just nii, nagu oli ette näidatud.

2.3 Bluetooth, BLE ja järjestikune andmevahetus

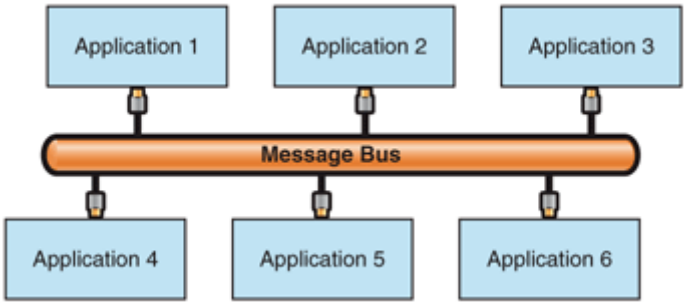
Bluetooth on liides seadmete omavaheliseks andmevahetuseks. Kasutatakse lühemate distantsinde peal traadita järjestikune andmevahetuseks kiirusega üldkasutusel olevates nutitelefonides kuni 25 Mbit/s [27].

Järjestikune andmevahetus (*serial communicatoin*) kasutab üht kanali andmete edastamiseks. Seega kogu informatsiooni edastatakse bitthaaval. Bluetooth'i puhul niisugune andmeedastus toimub RFCOMM protokolli järgi. Seda Bluetooth'i võib nimetada klassikaliseks Bluetooth versiooniks, kuna värkvõrkude ajal muutusid peamised vajadused selles tehnoloogias ning selle tulemusena tekkisid antud tehnoloogia uued versioonid. [26]

Mitteklassikaliseks versiooniks võib nimetada BLE ehk Bluetooth Low Enery versiooni Bluetooth tehnoloogiast, mille peamine eesmärk on töö väikse energiakuluga. Kõige tähtsam vahe antud töö rammes on selles, et BLE versioon ei luba suurte andmete vahetamist, mis võib takistada rakendute skaleerimist. [26]

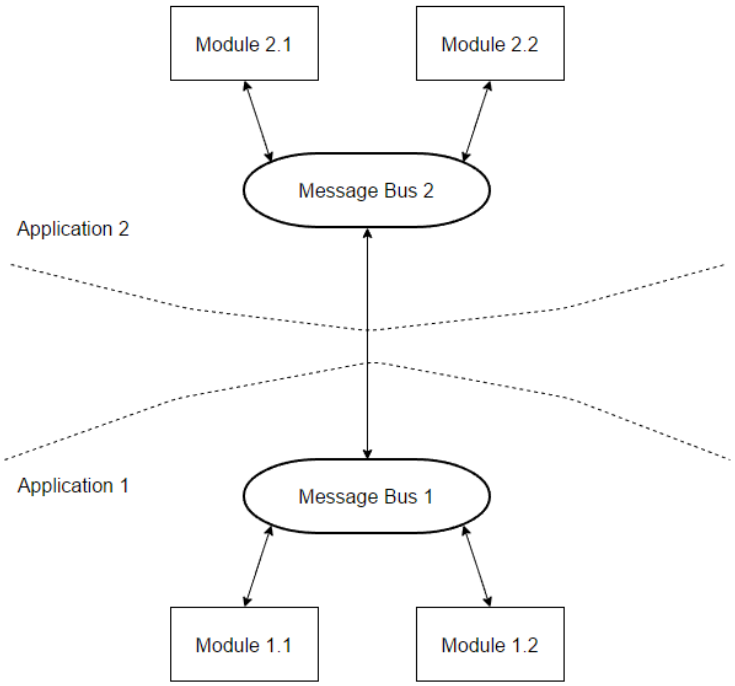
2.4 Message bus

Message bus või sõnumite siin on struktuur andmete vahendamiseks erinevate rakenduste või ühe rakenduse osade vahel. Struktuuri kasutamise idee on selles, et rakendus ei pea enam otseselt teiste rakendustega kontakteerima andmete vahetamiseks teiste rakendustega, vaid kasutab selleks vahendajat – sõnumite siini. Kui rakendus on saatnud sõnumite siini sõnumi, siis seda sõnumit saavad vastu võtta kõik ülejäänud sama sõnumite siiniga ühendatud rakendused. Kokkuvõttes – rakendus ise vastutab andmete saatmiste ja vastuvõtmise eest. [11]



Joonis 2. Andmevahetus läbi *message bus*'i [11]

Joonis 2 näitab klassikalist kasutamisviisi. Arendades seda struktuuri saab sõnumite siini funktsionaalsuse laiendamisega kasutada keerukamat struktuuri, Joonis 3 illustreerib seda. Seda struktuuri kasutatakse ka antud töö raames rakenduste loomiseks, vastavalt Raspberry server ja hübriidrakendus kasutavad enda sees esimest ja teist sõnumite siini ja vajadusel saadavad rakendused sõnumeid enda siinidest partnerrakenduse siini. Seda stuktuuri oleks raske skaleerida siinide suhtes. Antud juhul on seda struktuuri mugav praktikas kasutada kuna muutub ainult siinidega ühendatud moodulite arv.



Joonis 3. Keerukam struktuur sõnumite siinide põhjal

3 Hübriid mobiilirakenduse arendamine

Järgnevas peatükkis antakse ülevaadet hübriidmobiilirakenduse loomise protsessist, kirjeldatud on ka rakenduse komponentide ülesanded ja tööpõhimõtted. Kõik põhilised rakenduse protsessid on illustreeritud pildi või koodinäidisega.

3.1 Eeltöö

PhoneGap'i saab kasutada kahel erineval viisil: graafilise kasutajaliidesega või käsurea (CLI) rakendusena. Sisuliselt pakkuvad nad väga sarnast funktsionaalsust. Võrreldes GUI rakendustega, annavad käsurea rakendused parima ülevaate sellest, mis oli tehtud. GUI rakendustest saab harva näha tehtud tegevuste ajalugu, mis on väga tähtis projektide ehitamseks. Käsurea aknast saab näha nii kasutatud käske kui ka nende täitmise väljundit. Seetõttu valik oli tehtud käsurea rakenduse kasuks.

Arendamiskeskkonnaks oli valitud Visual Studio Code [12], kuna ta vaikimisi toetab JavaScripti ja Node.js keskkonda ning lubab igast kaustast teha iseseisvat projekti. Antud arendamiskeskkonda on sisseehitatud ka käsurea ja GIT'i pistikprogramm, mis annab võimaluse koodile tehtud muudatuste mugavaks ülevaatamiseks.

Hübriidrakenduse kompilleerimiseks oli valitud PhoneGap'i poolt pakutav veebiteenus PhoneGap Build [13]. Selle teenuse kasutamiseks on vaja Adobe ID [14] ehk kontod Adobe keskkonnas. Rakenduse kompilleerimiseks on vaja kas laadida veebirakendusele rakenduse projekt kausta *zip*[1] failina või määrata GIT repositoorimi[2], kust PhoneGap Build enne kompilleerimist hakkab koodi alla tõmmata.

Bluetooth funktsionaalsuse kasutamiseks PhoneGap hübriidrakendustes tuleb eelnevalt installeerida kolmanda isiku poolt arendatud pistikprogramme. Vaatamata sellele, et niisuguseid pistikprogramme on palju, nendest oli raske leida sellist, mis toetaks järjestikandmevahetust nii Android kui ka iOS platvormil, kasutades samal ajal

[1] *ZIP* – arhiivifaili nimilaiend

[2] Repositoorium – lähtekoodi hoidla

klassikalist Bluetooth ühendust. Ainukene pistikprogramm, mis sobis kõikide kriteerimite järgi, oli *cordova-plugin-bluetoothClassic-serial* [15].

Phonegap CLI ehk PhoneGap'i käsurea versiooni ja kõikide hübriidrakendusele vajalikke pistikprogrammide installeerimine toimub Node.js keskkonnas läbi *Node.js Package Manager*'i [16].

Lõppkokkuvõttes koosnes eeltöö protsess järgmistest sammudest:

1. Node.js allalaadimine ja installeerimine
2. PhoneGap CLI installeerimine NPM'i kaudu
3. PhoneGap projekti genereerimine Framework7 mustriga – see on üks vaikimisi pakutavatest võimalustest selleks, kuidas projekti genereerida
4. Sihtplatvormide määramine – Android ja iOS
5. Bluetooth pistikprogrammi installeerimine

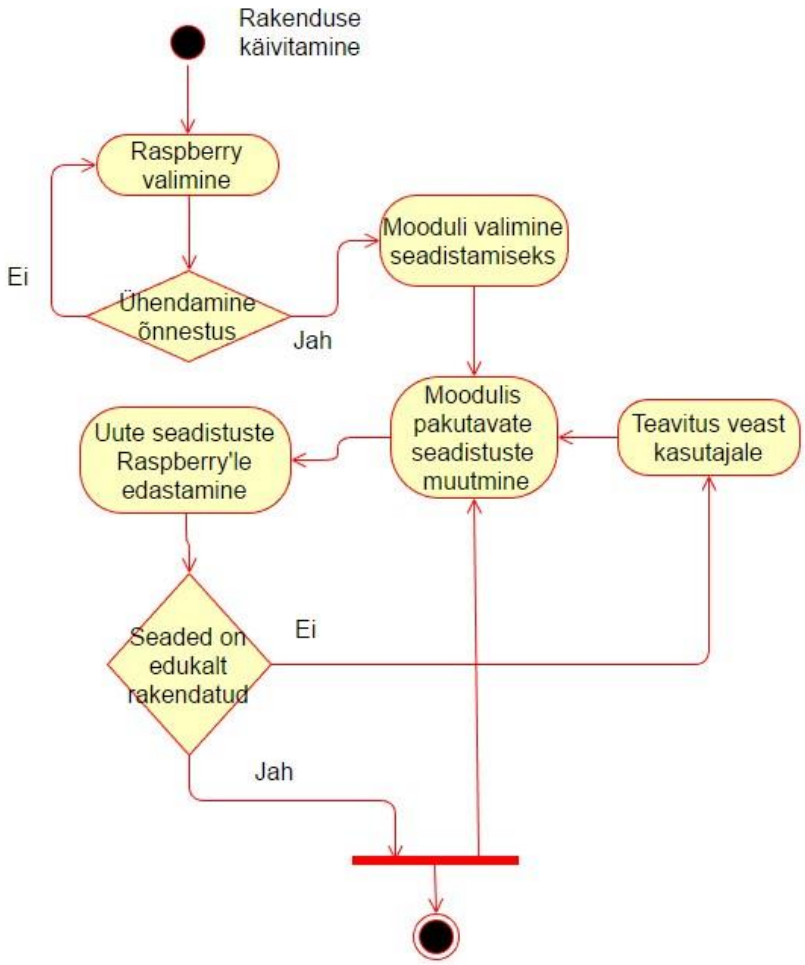
Phonegap CLI genereerib projekti kausta spetsiifilise arhitektuuriga, kus on nii kohustuslikud kui ka mittekohustuslikud hübriidrakenduse kompilleerimiseks vajalikud failid ja kaustad. Allpool kirjeldan kohustuslikke faile ning nende rolli:

- config.xml – fail. Peamine rakenduse konfigureerimisfail, sisaldab rakenduse meta-informatsiooni, näteks: rakenduse nimi ja autor, kasutatavate pistikprogrammide loetelu, ikoon, faili asukoht ja nõuded platvormile, kus rakendust jooksutakse.
- www – kaust. Sarnane veebiserveri kaustaga ning peab kindlasti sisaldama index.html faili, mida esimesena kuvatakse pärast rakenduse käivitamist. Peamise HTML faili nime saab muuta, aga sellele vastavalt peab olema korrigeeritud ka config.xml fail.

Tuleb pöörata tähelepanu sellele, et PhoneGap pistikprogramme installeerimise protsessis tekitatakse „plugin" kausta, kuid PhoneGap Build ignoreerib seda kausta ja laadib ise kompileerimise protsessiks vajalikke pistikprogramme config.xml failis oleva pistikprogrammide nimekirja järgi.

3.2 Arendusprotsess

Rakenduse funktsionaalsust oli otsustatud jagada kaheks vaadeks, esimene vaade on Raspberryga ühendamiseks, ja teine vaade Raspberry seadistamiseks. Teine vaade peab oma arhitektuuri poolt võimaldama lisaks WiFi seadistamisele teiste seadistamispistikprogrammide kasutamist. Rakenduse töö tegevusdiagrammil (Joonis 4) on kirjeldatud peamised tegevused.



Joonis 4. Mobiiliakenduse töö tegevusdiagramm

3.2.1 Rakenduse HTML struktuur

Projekti genereerimisel genereeritakse ka index.html fail koos töötava näidisrakendusega. Näidis juba kasutab Framework7 funktsionaalsust ning on vormistatud vastavalt raamistiku vajadustele, seega saab keskenduda veebilehe ehk hübriidrakenduse kasutajaliidese arendamisele. Kasutajatele nähatavatele objektidele HTML'is m määratakse *body* siltide sees. Framework7 kasutamine määrab oma reeglid ja *body* siltide

sees olev kood, arvestades arendatava rakenduse vajadusi, peab olema vormistatud järgmiselt:

```
...
<body>
  ...
  <!-- Vaaded -->
  <div class="views">
    <!-- Esimene vaade: Raspberry valimine paired seadmete hulgast -->
    <div class="view view-main">
      <!-- Navigatsioon -->
      <!-- Lehed -->
    </div>
    <!-- Teine vaade: seadistamis moodulid ja seadistamine -->
    <div class="view another-view">
      <!-- Navigatsioon -->
      <!-- Lehed -->
    </div>
  </div>
  <!-- Mustrid ja skriptid -->
  <script type="text/template7" id="mustriUnikaalneIdentifikaator">
    ...
  </script>
  ...
</body>
...
```

Joonis 5. Framework7 reeglite järgi *body* silti vormistus [7]

Peamise ehk esimese, mis rakenduse käivitamisel ette tuleb, vaate eesmärk on anda kasutajale võimalust valida *paired* seadmete[1] hulgast Raspberry't mida ta tahab seadistada. Kõik sobivad seadmed kuvatakse nupudena, kus nupu tekst on vastava seadme nimi. Nupu vajutamisel hakkab rakendus valitud seadmega ühendust looma. Ühenduse loomise jooksul kuvatakse kasutajale lehte laadimisanimatsiooniga.

Ülaltoodud funktsionaalsuse realiseerimiseks on mugav kasutada kahte lehte: esimene leht seadmete nimistuga ja teine laadimisanimatsiooniga. Bluetooth üheduse loomise protsessi katkestamise võimalust jätame antud töö raames välja. Selle funktsiooni välja jätmine tõstab rakenduse töökindlust. Seda on lihtne saavutada, kui lehtede vahel puudub navigatsioon. Sel juhul puuduks kasutajal võimalus ühenduse loomise ajal midagi valesti teha. Kuna ühendamise protsess ei võta tavaliselt rohkem kui 10 sekundit, siis see ei

[1] *Paired* seade – bluetooth pärast ühekordsed ühendamist seadmega jättab seda seadet meelde. Seda seadme nime kuvatakse *paired* seadmete nimistus. Idee järgi sobiks „partner seade" tõlgendus.

tekitaks ka kasutajal suurt segadust, kui aga võtab rohkem, siis ühendus lihtsalt katkestakse.

3.2.2 Raamistikute sündmused

Framework7 korraldab lehtede tööd nii, et kui nendevahelist navigeerimist ei ole, siis iga kord kui lehti ühe vaate raames vahetatakse, laaditakse uut lehte täiesti nullist. Igal lehel on olemas oma elutsükkel ning spetsiaalsed elutsükli sündmused. Üks kõige tähtsamatest lehe elutsükli sündmustest on *pageInit* sündmus, mis juhtub kohe, kui veebileht on täielikult kuvatud kasutajale. Pärast selle sündmuse juhtumist võib kindel olla, et kõik HTML failis kirjeldatud DOM elemendid eksisteerivad ning nendega saab manipuleerida JavaScripti abil.

PhoneGap annab samuti võimaluse juhtida rakenduse tööd vastavalt rakenduse elutsükli sündmustele. Sel juhul tähtsaks sündmuseks on *devicready* sündmus. Selle sündmuse juhtumine signaliseerib, et kõik PhoneGap hübriidrakendusega kaasnevad süsteemid (vaata Joonis 1) töötavad. Arendatava rakenduse puhul *devicready* sündmus lubab kasutada Bluetooth pistikprogrammi, mida läheb vaja kohe esimeses vaates *paired* seadmete leidmiseks ja nendega ühendamiseks.

Nii Framework7 kui ka PhoneGap raamistiku poolt määratud sündmusi on lihtne hallata, võttes neid eraldi. Juhul, kui kaks raamistikku töötavad koos, võib sündmuste haldamine olla tõsiselt raskendatud. Nii PhoneGap kui ka Framework7 dokumentatsioonis puudub informatsioon, kumb sündmus – *devicready* või *pageInit* – toimub varem. Selleks, et vältida sellega seonduvaid probleeme, kirjutasin JavaScriptis järgmise koodi:

```
...
$$(document).on('deviceready', function() {
    mainView.router.reloadPage('index.html');
    ...
});
...
```

Joonis 6. Kood initsialiseerimis sündmuste sünkroniseerimiseks

Antud kood on lühike, kuid tagab kogu rakenduse korrektset tööd. Esiteks tuleb mainida, et *pageInit* sündumus toimub iga kord lehe laadimisel ja samuti ka uuendamisel, kuna uuendamiseks kasutatud meetod laadib lehte täiesti nullist. Seega, vaatamata sellele, mis nendest kahest sündmustest juhtub varem – pärast *devicready* sündmust lehte uuendatakse. Lehe uuendamisega kindlustatakse, et *pageInit* toimub pärast *devicready*

sündmust. Selle probleemi lahendus oli kriitiline rakenduse jaoks, kuna pistikprogrammide kasutamine algab kohe esimeses vaates ning juhul, kui nad ei oleks enne lehe laadimist kättesaadavad, kasutaja ei näeiks ühtegi seadet *paired* seadmete nimistus.

3.2.3 Lehtede ja vaadete vaheline andmeedastus

Nii veebis kui ka antud juhul hübriidrakenduses - kogu JavaScript'i kood jookseb ühes keskkonnas seega võimalusi andmete edastamiseks ühest vaatest/lehest teisse on väga palju. Framework7 pakkub aga oma lahendust, mis hästi klappib kokku raamistiku ülejäänud funktsionaalsusega. Vaatleme järgmist näidet:

```
...
<script type="text/template7" id="deviceTemplate">
    <li>
        <a href="./components/loading.html?mac={{mac}}" class="...">
            {{name}}
        </a>
    </li>
</script>
...
```

Joonis 7. Seadme valimise nupu muster

Näide sisaldab kahte selle rakenduse raames tähtsat vahendit. Esiteks – antud koodi osa on muster, mida saab mitmekordselt kasutada JavaScript'is veebilehele samasuguste stiilidega elementide lisamiseks. Nende elementide vahel muutub ainult neile lisatud kontekst. Sõnad loogeliste sulgude sees tähendavad tegelikult muutujaid, mida väärtustatakse JavaScript'iga iga kord antud mustri viimistlimisel. Täieliku muutujate väärtuste komplekti nimetatakse kontekstiks. Mustri viimistlemise protsessis asendatakse kõik mustris olevad muutujad kontekstiga määratud väärtustega ja tagastatakse lõpliku HTML'i tavalise tekstina.

Teine vahend selles näites on URL parameetrid. *href* atribuut näitab, missugust lehte avab rakendus pärast *a* elementi vajutamist, kuid lisaks sellele pärast küsimärki URL'is on võimalik lisada muutujäid ja nende väärtusi. Niisugusel viisil edastatud muutujaid ja nende väärtust saab kasutada laaditud veebilehe JavaScript'il. Vaadeldaevat mustrit kasutatakse esimeses vaates *paired* seadmete nimistu genereerimiseks.

3.2.4 *Serial* ühenduse loomine

Eelmise peatüki koodi näidest (Joonis 7) oli näha, et pärast seadme valimist viib rakendus kasutajat *loading.html* lehele, andes kaasa valitud seadme MAC aadressi. Laadimislehe kasutajavaade sisaldab ainult lõpmatut laadimisanimatsiooni. Lehe initsialiseerimisel käivitatakse järgmist koodi:

```
...
var deviceId = e.detail.page.query.mac;
var interfaceIdArray = "00001101-0000-1000-8000-00805F9B34FB";
var connectionSucceeded;
...
function connectSuccess() {
    connectionSucceeded = true;
    $$('.view-main').hide();
    $$('.view-operations').show();
    ...
}
function connectionFailure(err) {
    ErrorHandler.showModalError(err);
    $$('.view-main').show();
    $$('.view-operations').hide();
    mainView.router.loadPage('index.html');
    ...
}
bluetoothClassicSerial.connect(deviceId, interfaceIdArray, connectSuccess,
connectionFailure);
setTimeout(function() {
    if (connectionSucceeded !== true) {
        connectionFailure('Device is not accessbile. Choose right Raspberry
or reboot it.');
    }
}, 10000);
...
```

Joonis 8. Raspberry'ga bluetooth *serial* ühenduse loomine

Koodilõigus (Joonis 8) esiteks salvestatakse muutujasse veebilehele URL parameetritega edastatud seadme MAC aadressi. Siis määratakse ühendatava liidese identifikaatori ja muutuja, mis on kasutatud kontrollimaks, et ühendus õnnestus 10 sekundi jooksul. Ühenduse loomiseks kasutatav Bluetooth pistikprogramm pakub võimalust määrata funktsiooni, mida kutsutakse välja juhul, kui ühendus ebaõnnestus.

Rakenduse prototüübi testimise jooksul tuli välja, et kasutatav pistikprogramm ei viska ühtegi veateadet olukorras, kus seadmega saab ühendust võtta, aga määratud liidesest vastust ei saa. Niisugune olukord juhtub siis, kui Raspberry töötab, aga Raspbery

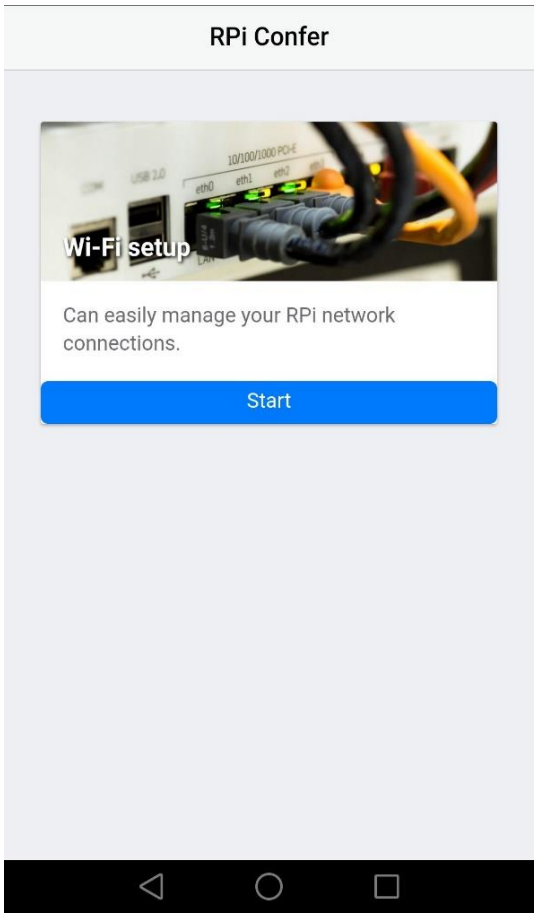
serverrakendus ei ole käivitatud. Olukorra lahendamiseks oli otsustatud kasutada funktsiooni, mida käivitatakse 10-sekundilise hilinemisega ühenduse loomise alguse hetkest. Seda oli võimalik teha tänu sellele, et *bluetoothClassicSerial.connect* meetod on asünkroonne. Juhul kui ühendus on õnnestunud, kutsutakse välja *connectSuccess* funktsiooni, mis annab *connectionSucceeded* muutujale *true* väärtust ja sel juhul tehtud kontrollfunktsioon 10 sekundi möödumisel ei tee midagi. Teistel juhtumitel kutsutakse sama *connectionFailure* meetodit, mida kutsub välja pistikprogrammi ühenduse ebaõnnestumisel.

Juhul, kui ühendus Raspberry'ga on õnnestunud, nagu on näha *connectSuccess* funktsioonist, vahetatakse vaadet. Rakenduse kasutajaliidese lihtsustamise mõttes oli otsustatud jätta ainult kaks olukorda, millal vaadet vahetatakse. Esimene on juba mainitud, lisaks sellele toimub vaadete automaatne vahetumine ühenduse katkestamisel. Ühenduse katkestamisel näidatakse kasutajale vastavat veateadet ja suunatakse esimesesse vaatesse.

3.2.5 Seadistamismooduli valimine

Rakenduste süsteemi idee on olla modulaarne. Iga moodul võib pakkuda unikaaltset funktsionaalsust Raspberry seadistamiseks. Näidisena on tehtud WiFi seadistamismoodul nii mobiili kui ka Raspberry serveri jaoks. Mooduleid oli otsustatud näidata nimekirjas kaartidega. Allpool on lisatud valmis vaate ekraanipilt (Joonis 9).

Ekraanipildil olev kaart on kirjeldatud otseselt lehe HTML koodis, aga vajadusel saab kõike mooduleid, nende kaarte ja koodi dünaamiliselt importida JavaScript'iga. Küll aga see vajaks lisa teeke või raamistike, näiteks nagu Browserify [17], mis omakorda tõstaks näidislahenduse keerukust. Edaspidiseks rakenduse arendamiseks ja uute moodulitega laiendamiseks aitaks dünaamiline import modulite siseandmete kaitsmisel, vähendaks rakenduse arhitektuurilist erosiooni.



Joonis 9. Seadistuste moodili valimise leht

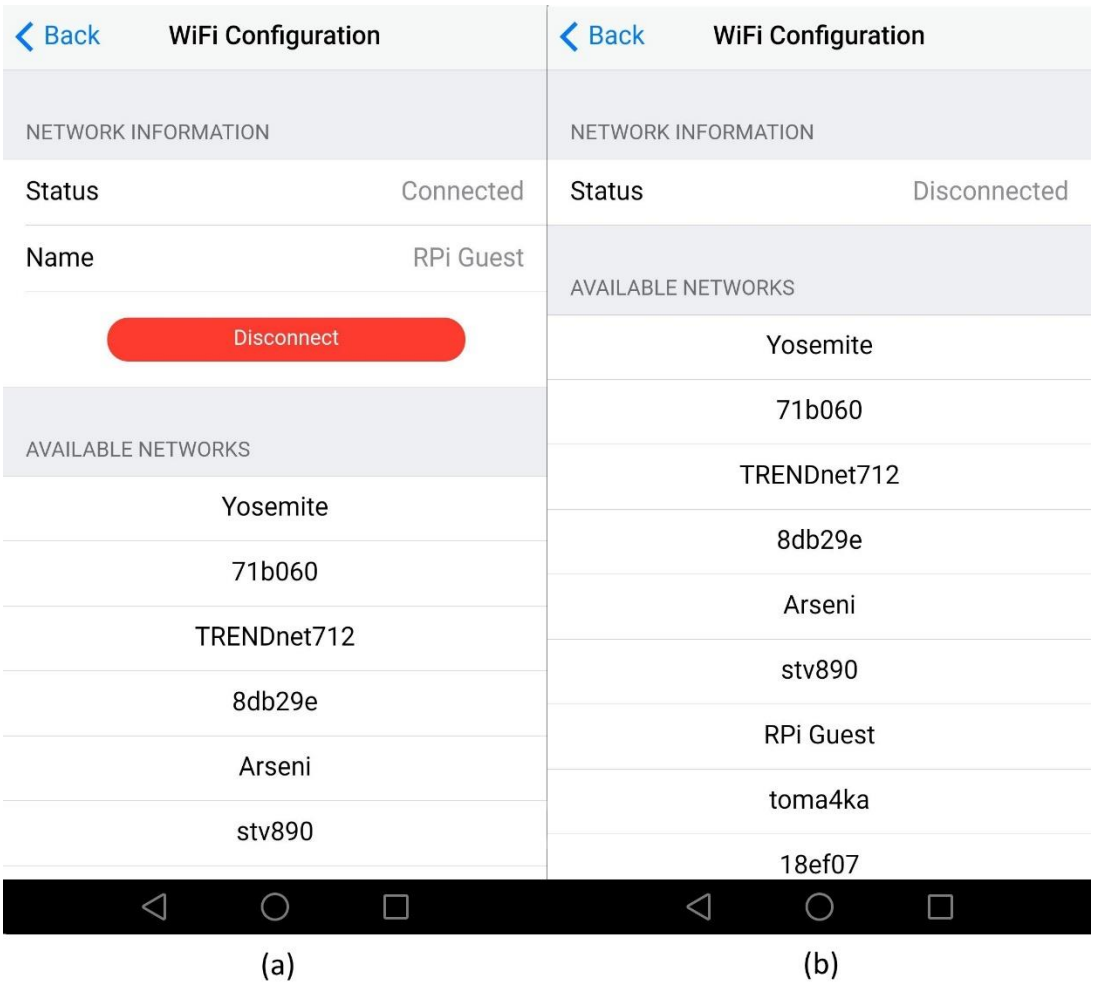
3.2.6 WiFi seadistamis moodul

Mooduli eesmärk on praktilisest küljest lihtsustada WiFi seadistamisprotsess, ja teoreetilisest küljest näidata ka kogu rakenduste süsteemi potentsiaali. WiFi seadistamis moodul pakub kasutajale nelja funtsiooni:

1. WiFi ühenduse hetkestaatuse näitamine - kas ühendus on või ei ole
2. Avalikult nähtavate WiFi võrkude nimekirja näitamine
3. Parooliga kaitstud WiFi võrguga ühendamine
4. WiFi võrguga ühenduse katkestamine

Kasutajaliidese disaini ideeks oli luua võimalikult head kasutajakogemust ja samal ajal säilitada lihtsust. Kõige olulisema info antud pistikprogrammi jaoks on otsustatud näidata kohe peamisel lehel. Pärast seadistamismooduli lehe äralaadimist näeb kasutaja Raspberry WiFi ühenduse staatust ja kättesaadevate võrkude nimekirja (Joonis 10). Juhul,

kui WiFi ühendus on juba seadistatud, näidatakse vastava staatuse ja seadistatud võrgu nime ning nuppu ühenduse katkestamiseks. Vastasel juhul näidatakse ainult WiFi ühenduse staatust "*disconnected*".



Joonis 10. WiFi seadistamis mooduli peamine leht: (a) WiFi on seadistatud, (b) WiFi ühendust pole

Mooduliga pakutav funktsionaalsus vajab Raspberry'ga suhtlemist. Rakendus kasutab kahepoolseks andmete vahetamiseks Bluetooth pistikprogrammi *subscribeRawData* ja *write* meetodeid. Esimene mainitud meetud on mõeldud selleks, et Bluetooth *serial* liidese pealt andmeid oodata ja andmete kättesaamise puhul edastada neid töötlemisfunktsioonile. Teine mainitud meetod on mõeldud andmete saatmiseks. Mõlemad meetodid eeldavad, et Bluetooth *serial* ühendus on eelnevalt loodud. Mõlemate meetodite kasutuskontekst on illustreeritud allolevas koodilõigus.

Kuna mõlemad rakendused kasutavad andmete hoidmiseks JSON formaati, siis nii enne andmete saatmist kui ka pärast andmete kättesaamist tuleb konverteerida andmeid vastavalt kas tavaliseks tekstiks või JSON formaadiks.

```
...
$$(document).on('deviceready', function() {
    ...
    MessageBus.setupBasicSender(function(data) {
        var interfaceIdArray = "00001101-0000-1000-8000-00805F9B34FB";
        bluetoothClassicSerial.write(interfaceIdArray, JSON.stringify(data),
function() {
                    console.log(JSON.stringify(data));
             }, ErrorHandler.showModalError);
    });
    MessageBus.subscribeModule('wifi');
});
...
$$(document).on('pageInit', '.page[data-page="loading"]', function(e) {
    ...
    function connectSuccess() {
        ...
        bluetoothClassicSerial.subscribeRawData(interfaceIdArray,
function(data) {
            const bytes = new Uint8Array(data);
            const string = new TextDecoder("utf-8").decode(bytes);
            const parsedData = JSON.parse(string);
            MessageBus.receive(parsedData);
            bluetoothClassicSerial.clear(interfaceIdArray);
        }, ErrorHandler.showModalError);
    }
}
...
```

Joonis 11. Kood mis haldab Raspberry ja hübriidrakenduse suhtlemist

Koodi esimeses osas ehk *deviceready* sündmuse haldamise funktsioonis on kasutatud *MessageBus* andmestruktuuri. Antud struktuur on spetsiaalselt käesoleva bakalaureusetöö raames arendatud, et lihtsustada rakendustevahelist suhtlemist. Sama struktuuri kasutatakse suhtlemiseks nii hübriidmobiilirakenduses kui ka Raspberry serverrakenduses. Struktuuri teoreetiline idee on kirjeldatud peatüki „2.4 Message bus“ lõpus. Struktuuri võiks kirjeldada nagu klassikalist *MessageBus*’i, mis on adapteeritud projekti vajadustele. Lähemalt struktuuri vaadetakse „MessageBus andmestruktuur“ peatükis, mis asub 34 lehel.

Eelseadistused, mis on tehtud ülaltoodud koodilõigus, on oma ideede poolt järgmised:

1. Määratakse funktsiooni, mida *MessageBus* hakkab kasutama sissetulevate andmete edasi saatmiseks

2. WiFi seadistamis moodulit registreeritakse *MessageBus*'is lühendatud nimega "*wifi*".

3. Määratakse funktsiooni, mida automaatselt käivitakse uute andmete sissetulemisel.

Mainitud eelseadistused aitavad korraldada aja mõttes efektiivset andmevahetust ning annavad võimaluse nii WiFi kui ka potentsiaalsetele järgnevatele moodulilete lihtsustatud viisil andmeid saata ning moodulile ettenäidatud andmeid kätte saada. Näiteks võib tuua mooduli peamise lehe koodi (Joonis 12).

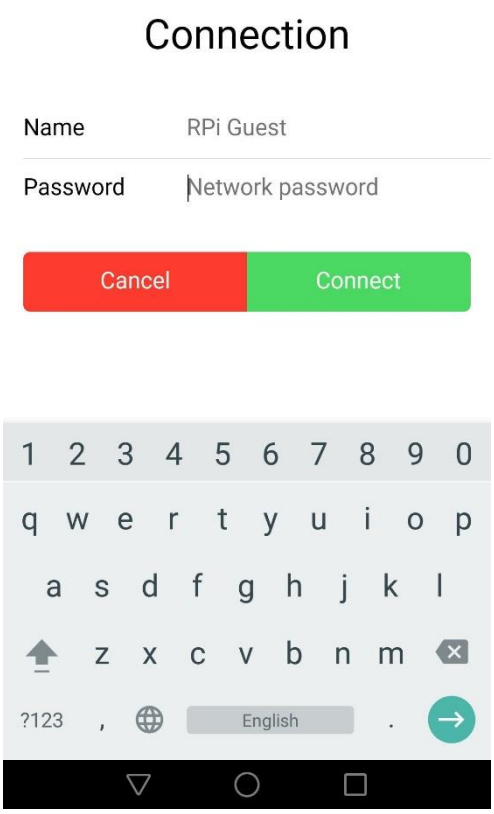
```
...
$$(document).on('pageInit', '.page[data-page="config-wifi"]', function(e) {
    var wifiSubscription = MessageBus.subscribers['wifi'];

    wifiSubscription.on('wifi_list', function(data) {
        const wifiNetworks = data.networks;
        $$('#wifiaps').html('');
        wifiNetworks.forEach(function(network) {
            $$('#wifiaps').append(Template7.templates.wifiApTemplate({ ssid:
network.ssid }));
        });
    });

    wifiSubscription.on('wifi_state', function(data) {
        const network = data;
        // Connected või Disconnected
        $$('#nstat').html(capitalizeFirstLetter(network.connection));
        // Edasi on ülejäänd kood andmete kuvamiseks veebilehele
        ...
    });

    wifiSubscription.send('wifi_state', {});
    wifiSubscription.send('wifi_list', {});
});
...
```

Joonis 12. WiFi mooduli peamise lehe kood

Antud koodi võib nimetada lakooniseks, kuna JavaScript'i mõttes antud lehe initsialiseerimisprotsessis täidetakse sisuliselt 5 käsku. Esiteks saadakse MessageBus'ist vajalikku objekti, siis seadistakse kahte funktsiooni, mis hakkavad Raspberry vastust

töötlema, ja lõpuks saadetakse 2 päringut Raspberry serverrakendusele. Päringuga edastatakse antud juhul tühja andmete objekti.

Viimane, mainimata jäänud leht arendatavas mobiilirakenduses, on WiFi üheduse seadistamisleht, kus kasutaja saab sisestada WiFi parooli ja teostada ühendust ühe nupuvajutusega või minna tagasi WiFi seadistamismooduli kodulehele. WiFi'ga ühenduse seadistamiseks tuleb eelnevalt valida sobiva võrgu, selleks tuleb kasutajal vajutada vastava võrgu nimega nuppu (Joonis 10 – iga "*Available Networks*" nimestukus olev element on nupp) kättesaadevate võrkude nimestikus. Ühenduse seadistamisleht avaneb pärast valiku tegemist. Lehe kasutajaliidest illustreerib Joonis 13. Võrgu nimi (*Name*) lehel täidetakse automaatselt vastavalt peamisel lehel valitud võrgule, kasutaja ei saa seda antud lehel muuta. Parooli (*Password*) välja täidab kasutaja.



Joonis 13. Leht WiFi üheduse seadistamiseks

JavaScript koodi osas antud leht sarnaneb mooduli peamise lehe koodiga. Samamoodi saadakse kätte WiFi seadistamismoodulile vastava *MessageBus* objekti, mida edaspidi kasutatkse ühenduse loomise päringu saatmiseks ja vasutust töötleva funktsiooni määramiseks. Vahe on selles, et antud moodul lisab päringule ka andmeid, mida on vaja WiFi ühenduse loomiseks. Saadetavad andmed on võrgu nimi ja parool. Vastusena

Raspberry serveri poolt tuleb informatsioon, kas ühendus õnnestus või mitte. Vaatamata ühendusprotsessi tulemusele, kasutajat suunatakse tagasi mooduli pealehe, mis on uuendatud ja näitab praeguse ühenduse staatust. Kui ühendus WiFi võrguga ei õnnestunud, enne eelmisele lehele suunamist näidatakse kasutajale vastavat veateadet.

4 Raspberry serverrakenduse arendamine

Järgnevas peatükis antakse ülevaadet Raspberry'le rakenduse loomise protsessist. Võrreldes hübriidmobiilirakenduse loomise protsessiga, siin on kasutatavate tehnoloogiate hulk vähem.

4.1 Eeltöö

Arendamiseks vajalik riistvara, Raspberry Pi 3 Model B ja MicroSD[1] kaart, oli olemas. Töö alustamiseks oli vaja installeerida operatsioon süsteemi mälukaardile. Raspbian Lite süsteemi on võimalik Raspberry Pi ametlikult veebilehelt allalaadida. Selle süsteemi mälukaardile panemiseks tuli kasutada Etcher tarkvara [19]. Etcher esiteks formateerib valitud mälukaarti vastavalt installeeritava operatsioonisüsteemi vajadustele ja teiseks teostab installeerimist.

Kasutajaliideseta süsteemidega on mugav töötada kasutades SSH protokolli, mis võimaldab luua kahe arvuti vahele turvalise kanali andmevahuseks [20]. SSH protokolli saab kasutada erinevatel eesmärkidel, aga antud töö raames kasutatakse seda Raspberry sisselogimiseks ja kaugsedasitamiseks. Siinkohal on aga oluline mainida, et kuni 2016. aasta novembrini oli Raspbian süsteemidel vaikimisi SSH server sisse lülitatud, ehk kohe pärast süsteemi jooksutamist oli võimalik sinna SSH kaudu sisse logida. Hiljem oli SSH server kõikides uuemates Raspbian versioonides vaikimisi välja lülitatud. Selleks, et seda sisse lülitada, on vaja vaid panna mälukaardi juurkausta üks tühi fail nimega „ssh" [21].

Pärast ülalkirjeldatud operatsioonide teostamist pannakse mälukaart Raspberry'sse ning ühendatakse Raspberry vooluga ja traadi abil võrguga. Edaspidiseks seadmiseks on vaja leida Raspberry *Internet Protocol*[2] aadressi, mis on unikaalne iga seadme jaoks ühe võrgu raames. Kodustes tingimustes kasutatakse tavaliselt ühte sisevõrku ja kõik ühe ruuteriga seotud seadmed asuvad ühes võrgus. Raspberry aadressi leidmiseks on mitu võimalust. Kui ruuterisse on võimalik sisse logida, sealt saab tavalisest kätte ka ühendatud seadmete

[1] MicroSD – micro variant (füüsiliste mõõtmete mõttes) Secure Digital kaardist mis on välkmäluga, mälukaardi format [18]

[2] Internet protocol (lühidalt IP) – internetti protokoll

andmeid. Kui ruuterisse sisselogimise võimalus puudub, saab spetsiaalse tarkvaraga, nagu näiteks nmap[1], võrku skaneerida.

Pärast Raspberry aadressi kättesaamist tuleb luua sellega SSH ühendus. Töö raames oli selleks kasutatud MobaXterm tarkvara tasuta versioon, mille eeliseks on SSH sessioonide salvestamise võimalus [22]. SSH ühenduse loomiseks on vaja määrata sihtmasina aadresi, kasutajanime ja parooli. Parooli võib vahele jätta, seda küsitakse terminal aknas eraldi juhul, kui seda on antud kasutaja jaoks vaja

SSH üheduse olemasolul saab süsteemi seadistama hakata. Anutd juhul koosnes seadistamisprotsess kahest etapist: süsteemi komponentide uuendamine ja vajaliku tarkvara installeerimine. Näitab mis käskudega oli süsteem seadistatud.

```
sudo apt-get update
sudo apt-get upgrade
curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -
sudo apt-get install -y git nodejs build-essential libbluetooth-dev
```

Joonis 14. Raspbian süsteemi seadistamis käsud

Kaks esimest käsku on süsteemikomponentide uuendamiseks, kolmas käsk lisab süsteemi paketide allikateks uusi allikaid. [23] Viimane käsk installib GIT, Node.js tarkvara ning kahe süsteemi teeke, mida on vaja Bluetooth NPM mooduli jaoks.

4.2 Arendusprotsess

Raspberry serverrakendust oli otsustatud jagada kolmeks osaks ning jagada neid erinevatesse failidesse/kaustadesse. Kuna arendamiseks kaasutatud Nodej.js keskkond pakub võimalust moodulite eksportimiseks ja importimiseks – rakenduse osadeks jagamine ei vaja seega lisaraamistikke või teekide kasutamist. Rakenduse osad koos nende põhiülesannetega on järgmised:

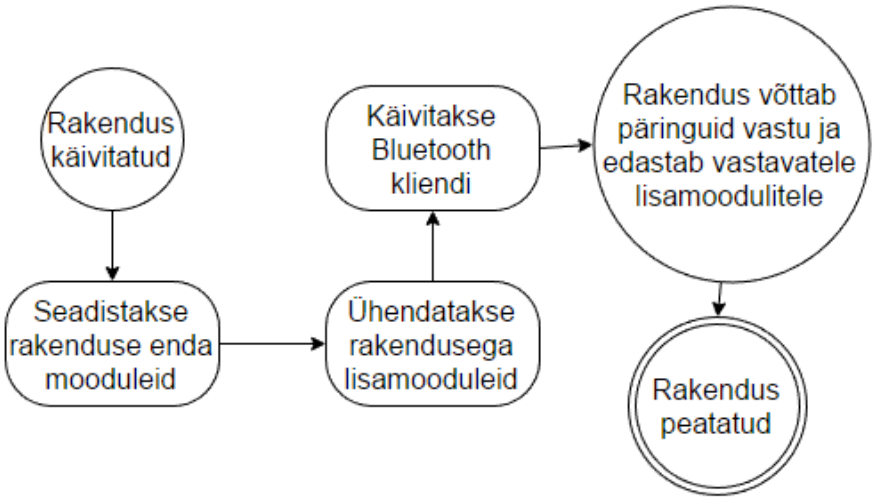
- Peamine rakenduse osa - praktiliselt üks fail, mis paneb rakenduse teisi osi kokku ning ühendab neid ühe informatsioonivõrgu sees. Haldab Bluetooth'i suhtlemisprotsessi ning andmete ja päringute liikumist rakenduse sees.

[1] Nmap - tasuta tarkvara võrgu analüüsimiseks

- Rakenduse enda moodulite osa – sisaldab kõiki (välja arvatut NPM) mooduleid, mida vajab peamine rakenduse osa oma tööks. Eesmärk on lihtsustada peamise osa koodi ja seega tõsta ka selle loetavust.

- Rakenduse lisamoodulite osa – sisaldab mooduleid, millega hakkab nutitelefoni rakendus suhtlema. Lisamoodulite ühendamine peaosaga tomub dünaamiliselt, ehk uue mooduli lisamiseks tuleb ainult vastava kausta luua ning paigaldada sinna mooduli koodi.

Allolev pilt illustreerib rakenduse elutsükli olekuid ning seda, mis toimub olekute vahetamisel. Täpsustamiseks tahaks mainida, et rakenduse enda sees ei ole loogikat mis teadlikult peatab rakenduse tööd, kuid see võib juhtuda, kui mõne funktsiooni käivitamisel leiab aset erandjuhtum.



Joonis 15. Raspberry server rakenduse elutsükkel

Rakendusi mis kasutavad või võimaldavad kasutada mitmekihelist arhitektuuri on lihtsam skaleerida võrreldes rakendustega kus kogu funktsionaalsus on kirjeldatud ühes failis. Antud töö raames oli otsustatud, et MessageBus andmestruktuuri ja lisamoodulite laadijat tuleb eraldada rakenduse peamisest osast ja realiseerida moodulitena. Antud juhul peamine rakendus kasutab vaid oma moodulite funktsionaalsust ning vajadusel saab oma moodulite seadistusi muuta, näiteks lisamoodulite kausta nime või asukohta vahetada. Lisaks sellele tõstis moodulite kasutamine tunduvalt rakenduse peaosa koodi loetavust. Kogu rakendus mahtus 24 koodireale.

Moodulite laadimine toimub Node.js sisseehitatud funktsioonide abil:

```
var server = new(require('bluetooth-serial-
port')).BluetoothSerialPortServer();
var MessageBus = require('./modules/message-bus');
MessageBus.setupBasicSender(function(data) {
    server.write(new Buffer(JSON.stringify(data)), function(err,
bytesWritten) {
        if (err) console.log('Error!');
        console.log('Response with ' + JSON.stringify(data).substring(0, 100)
+ '...');
    });
});
var appModules = require('./modules/module-importer')(MessageBus);
...
```

Joonis 16. Moodulite importimine peamises rakenduse osas

Kõikide moodulite sisselaadimine toimub sünkroonselt, seega näiteks ülaloleva koodinäite viimasel real kasutatud sõltuvuse sisestamine (*Dependency Injection*) edastab sisselaaditavale moodulile valmis objekti. Viimase rea moodul on ka rakenduse raames omapärane. Selle ülesandeks on teatud kaustast laadida peamisesse programmi kõik lisamoodulid.

Lisamoodulite kaustas peab iga lisamooduli jaoks olema üks kaust. Selles kaustas peab olema main.js fail. Lisamoodulite laadimisel kasutatakse nende kaustade nimesid moodulite nimedena ning laaditakse eelnevalt mainitud faili, mis peaks lisamooduli funktsionaalsust kokkuvõtma. Eraldiseisvate lisamoodulite tööprotsessi ei mõjuta moodili nimi (kui seda ei vaja moodul ise), kuid mooduli nime on vaja lisamoodulitevalisteks andmevahetuseks. Antud töös on realiseeritud ainult WiFi lisamoodul ja seega moodulite omavalise suhtlemist ei ole võimalik kasutada, kuid rakenduse skaleerimisel see võib oluliselt lainedada lisamoodulite võimalusi.

Bluetooth NPM repositooriumis leitud moodul *bluetooth-serial-port* võimaldas peamise rakenduse tööd organiseerida sarnaselt hübriidmobiilirakenduse koodiga (Joonis 17), ainukese vahega, et antud rakedus ei loo ühendusi ise, vaid võtab kõiki ühendusi vastu. Vaatamata sellele, et rakendus võtab vastu ükskõik mis andmeid, töödeldatkse ainult rakenduse spetsiifilisele mustrile vastavaid andmeid.

```
...
server.listen(function(clientAddress) {
    console.log('Client: ' + clientAddress + ' connected!');
}, function(error) {
    console.error("Something wrong happened!:" + error);
});

server.on('data', function(jsonString) {
    try {
        var jsonMessage = JSON.parse(jsonString);
        MessageBus.receive(jsonMessage);
    } catch (err) {
        console.log(err);
    }
});

server.on('closed', function() {
    console.log("Client closed connection!");
});
```

Joonis 17. Raspberry server rakenduse peamise osa kood

4.2.1 MessageBus andmestruktuur

MessageBus on üks kõige tähtsamatest andmestruktuuridest kogu rakenduste süsteemis. Tänu sellele, et nii hübriidmobiilirakendus kui ka Raspberry serverrakendus kasutavad JavaScript'i – täpselt sama andmestruktuuri on kasutatud mõlemates rakendustes. Andmestruktuuri kood on toodud antud töö „Lisa 1" osas.

Antud andmestrutuuri kasutamise peamine eesmärk on kiirendada andmete vahetust rakenduste vahel. Seda on saavutatud *on* ja *receive* meetoditega. *Receive* meetod on mõeldud kasutamiseks rakenduse peaosas. Seda meetodit käivitatakse siis, kui andmed teisest seadmest on täielikult kättesaadud ning neid on juba võimalik vastava moodulile töötlemiseks edastada. *On* meetod on mõeldud selleks, et määrata funktsiooni konkreetse päringu töötlemiseks.

Andmete saatmisprotsess võib olla ja on antud juhul erinev erinevate rakenduste puhul. Kuid iga moodul peab olema võimeline enda nimest päringuid saatma. Selleks on tehtud meetodid *setupBasicSender*, *send* mis on põhiobjekti sees ja *send* mis on *subscribers* hulgas iga objekti sees. Nendest meetoditest lähemalt:

- *setupBasicSender* –kirjutab ümber põhiobjekti *send* meetodit teise meetodiga, mis peab olema ette antud *setupBasicSender* arugumendis.

- *send* põhiobjektis – vaikimisi kirjutab argumendis etteantud andmeid konsooli
- *send* mis asub *subscribers* objektis – lisamoodulite poolt kasutatav meetod. Lisab automaatselt saadetavate andmetele mooduli nime, kust andmeid saadetakse.

Kuna iga mooduli nimi on selle mooduli kausta nimi ja kõikide moodulite kaustad asuvad ühes kaustas, siis olukord, kus andmeid edastakse valele moodulile, on välistatud. Tuleb ainult pöörata tähelepanu, et nii Raspberry serverrakenduses kui ka mobiilirakenduses oleksid moodulid sama nime all.

4.2.2 WiFi lisamoodul

Kõik lisamoodulid saavad sõltuvuse sisestamise (*dependency injection*) kaudu moodulite laadijalt objekti. Seda objekti kasutatakse nii päringute vastuvõtmise funktsioonide registreerimiseks kui ka päringute saatmiseks. Mõlemal juhul koos päringuga edastatakse andmeid, kuid andmete objekt võib olla tühi. Nii selle kui ka tulevate lisamoodulite töö on korradatud selle objekti ümber. Antud objekti saab iseloomustada inglise keelse sõnaga *subscription*[1].

Allpool on WiFi lisamooduli ülesehitust illustreeriv koodinäide (Joonis 18). Koodinäidisest on välja lõigatud ainult koodi osad, mis oma idee poolest dubleerivad näidises toodud koodi. Nagu on näha, läbi *subscription* objekti meetodite teostatakse andmevahetust mobiilirakendusega, aga andmevahetus ise on moodulist peidetud ehk sisuliselt toimub MessageBus'i kapseldamine.

WiFi seadistamiseks oli kasutatud NPM moodul „*wifi-control*“ [25]. Moodul pakub põhifunktsionaalsust WiFi võrkudega tegutsemiseks. Kõikide operatsioonide tulemuseks on JSON formaadis vormistatud andmed, mis omavad informatsiooni sellest, kas opertasioon on õnnestund või mitte; inglise keeles operatsiooni staatust kirjeldatavat sõnumit; andmeid, mis vastava operatsiooni tulemusena peaksid tekkima.

[1] Subscription – otsene tõlge infotehnoloogia konteksti raames eesti keeles puudub. Võib kirjeldada nagu kokkuleppet objektide edastamiseks, edastamine võib toimuda ka mõlemates suunas.

```
var wifi = function(subscription) {
    var WiFiControl = require('wifi-control');
    WiFiControl.init({
        debug: false,
        iface: 'wlan0',
        connectionTimeout: 10000 // in ms
    });
       ...
    subscription.on('wifi_list', function(data) {
        WiFiControl.scanForWiFi(function(err, response) {
            if (err) console.log(err);
            subscription.send('wifi_list', response);
        });
    });

    subscription.on('wifi_connect', function(data) {
        WiFiControl.connectToAP(data.ap, function(err, response) {
            if (err) console.log(err);
            subscription.send('wifi_connect', response);
        });
    });
       ...
}

module.exports = function(subscription) {
    wifi(subscription);
}
```

Joonis 18. WiFi lisamooduli koodi osa

5 Kokkuvõte

Bakalauerusetöö peamiseks eesmärgiks oli teha kaks rakendust. Üks rakendus on Raspberry Pi 3 Model B arvutile ja teine rakendus on Adobe PhoneGap raamistikus ehitatud mobiilpratfomidele hübriidrakendus. Mõlemate rakenduste jaoks arendada WiFi seadistamismooduli. Pärast nutitelefoni Raspberry'ga ühendamist läbi Bluetooth'i, peab olema kasutajal võimalus seadistada WiFi Raspberry peal.

Bakalaureusetöö tulemusena valmud rakendused vastutavad kõikidele üleval püstitud eesmärkidele. WiFi seadistamis protsess Raspberry jaoks on lihtsustatud ning ei erine oma keerukuse poolt analoogse protsessiga Android süsteemide sees. Lisaks sellele, raknedused on oma arhitektuuri poolt tehtud modulaarseteks ja seega toetab lisamoodulite kiir juurdepaneku. Raspberry ja moobilrakenduse moodulite omavahelise suhtlemise korraldamiseks oli ehitatud spetsiaalne andmestruktuur, mis aitas korradada andmevahetust effektiivsemal viisil. Andmestruktuur on unvisersaalne ja edaspidi võib olla kasutatud ka teistes projektides.

Kasutatud kirjandus

[1] Raspberry Pi 3 Model B specs [WWW] https://socialcompare.com/en/review/raspberry-pi-3 (20.05.2017)
[2] Raspbian Lite [WWW] https://www.raspberrypi.org/downloads/raspbian/ (20.05.2017)
[3] Raspbian [WWW] https://en.wikipedia.org/wiki/Raspbian (20.05.2017)
[4] Debian versioonide ajalugu [WWW] https://en.wikipedia.org/wiki/Debian_version_history (20.05.2017)
[5] Debian APT [WWW] https://wiki.debian.org/Apt (20.05.2017)
[6] Apache Cordova ja Adobe PhoneGap'i erinevus [WWW] http://www.informit.com/articles/article.aspx?p=2478076 (20.05.2017)
[7] Framework7 [WWW] https://framework7.io/ (20.05.2017)
[8] Käsper, A. HTML 5 tehnoloogia eelised ja puudused mobiilirakenduste arendamisel Adobe PhoneGap'i näitel: bakalaureusetöö. Tallinn, Tallinna Tehnikaülikool, 2016.
[9] PhoneGap rakenduse arhitektuur [WWW] http://www.justinshield.com/2014/05/cross-platform-mobile-development-phonegap-vs-xamarin/ (20.05.2017)
[10] DOM [WWW] https://www.w3schools.com/js/js_htmldom.asp (20.05.2017)
[11] Message Bus [WWW] https://msdn.microsoft.com/en-us/library/ff647328.aspx (20.05.2017)
[12] Visual Studio Code [WWW] https://code.visualstudio.com/ (20.05.2017)
[13] PhoneGap Build [WWW] http://docs.phonegap.com/phonegap-build/ (20.05.2017)
[14] Adobe ID [WWW] https://accounts.adobe.com/ (20.05.2017)
[15] Bluetooth Serial Classic [WWW] https://www.npmjs.com/package/cordova-plugin-bluetoothClassic-serial (20.05.2017)
[16] Node.js Package Manager [WWW] https://www.npmjs.com/ (20.05.2017)
[17] Browserify [WWW] http://browserify.org/ (20.05.2017)
[18] MicroSD [WWW] https://et.wikipedia.org/wiki/Secure_Digital (22.05.2017)
[19] Etcher [WWW] https://etcher.io/ (22.05.2017)
[20] SSH [WWW] https://et.wikipedia.org/wiki/Turvakest (22.05.2017)
[21] Raspbian SSH serveri käivitamine [WWW] https://raspberrypi.stackexchange.com/questions/40689/cannot-connect-to-raspbian-jessie-lite-but-to-raspbian-jessie (22.05.2017)
[22] MobaXterm [WWW] http://mobaxterm.mobatek.net/ (22.05.2017)
[23] Node.js installeerimine [WWW] https://nodejs.org/en/download/package-manager/ (22.05.2017)
[24] Bluetooth NPM moodul [WWW] https://www.npmjs.com/package/bluetooth-serial-port (22.05.2017)
[25] WiFi-control NPM moodul [WWW] https://www.npmjs.com/package/wifi-control (22.05.2017)
[26] Bluetooth vs Bluetooth Low Energy [WWW] https://www.link-labs.com/blog/bluetooth-vs-bluetooth-low-energy (22.05.2017)

[27] Bluetooth [WWW] https://en.wikipedia.org/wiki/Bluetooth (22.05.2017)

Lisa 1 – MessageBus andmestruktuur

```
1.  var MessageBus = {
2.      subscribers: {},
3.      subscribeModule: function(moduleName) {
4.          this.subscribers[moduleName] = {
5.              subscriptions: {},
6.              on: function(requestName, callback) {
7.                  this.subscriptions[requestName] = callback;
8.              },
9.              send: undefined
10.         };
11.         var basicSender = this.send;
12.         this.subscribers[moduleName].send = function(request, data) {
13.             basicSender({ module: moduleName, request: request, data: data });
14.         };
15.
16.         return this.subscribers[moduleName];
17.     },
18.     send: function() {
19.         console.log(JSON.stringify(data));
20.     },
21.     setupBasicSender: function(newSenderFunction) {
22.         this.send = newSenderFunction;
23.     },
24.     receive: function(message) {
25.         this.subscribers[message.module].subscriptions[message.request](message.data);
26.     }
27. }
```