TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Airiin Kadakmaa

ANDROIDI-PÕHISE KASUTAJALIIDESE LOOMINE PROGRAMMEERITAVALE TOITEPLOKILE

Bakalaureusetöö

Juhendaja: Meelis Antoi
magistrikraad
Kaasjuhendaja: Priit Pikk
bakalaureusekraad

Tallinn 2020

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: Airiin Kadakmaa

18.05.2020

Annotatsioon

Antud bakalaureusetöö eesmärk on luua kasutajaliides programmeeritavale toiteplokile, et võimaldada selle juhtmevaba kontrollimine. Eesmärgi saavutamiseks loob autor rakenduse, mis töötab mobiilsetel Androidi seadmetel.

Analüüsi käigus võrdleb autor olemasolevaid programmeeritavate toiteplokkide kasutajaliideseid, kirjeldab loodava rakenduse põhilisi nõudeid, mõttekäike protsessidest ja valib ülesande lahendamiseks sobivaimad töövahendid. Töö praktilises osas kirjeldab autor prototüübi disaini, selle põhjal rakenduse arenduse esimest etappi ning esialgset testimist.

Bakalaureusetöö tulemusena loob autor algse Androidi rakenduse, milles pannakse paika rakenduse struktuur ning tingimustest sõltuva andmete teisendamise ja kuvamise loogika.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 25 leheküljel, 5 peatükki, 23 joonist.

Abstract

Creating an Android Based User Interface for a Programmable Power Supply

The aim of this bachelor's thesis is to create a user interface for a programmable power supply, so that it may be controlled wirelessly. In order to reach that goal, an application capable of running on mobile Android devices is created by the author.

In the analysis part, the author compares existing user interfaces for programmable power supplies, describes the main requirements of the application being created, thought processes behind activities and chooses the tools best fitted for the task. In the practical part of the thesis, the author describes the design of the prototype, the first stage of application development based on the design and initial testing.

As a result of this bachelor's thesis, the author creates a tentative Android application, through which the application structure is set, as well as the logic behind data conversion and displaying.

The thesis is in Estonian and contains 25 pages of text, 5 chapters, 23 figures.

Lühendite ja mõistete sõnastik

agiilne arendus	Tarkvaraarenduse metodoloogiate grupp, mis põhineb iteratiivsel arendusel, kus nõuded ja lahendused arenevad koostöös tiimidega, kus rollid vahetuvad ja tööjaotus toimub spontaanselt [1]
Android	Operatsioonisüsteem nimega “Android”
Android-seade	Seade, millel jookseb operatsioonisüsteem Android
BLE	*Bluetooth Low Energy* – madala energiakuluga Bluetooth
Bluetooth	Tehnoloogia, mis võimaldab seadmetel juhtmevabalt üle lühikese vahemaa andmeid saata [2]
broadcaster	Bluetooth GAP saatja roll
CC	*Constant current* – püsiv voolutugevus
central	Bluetooth GAP keskne roll
CV	*Constant voltage* – püsiv pinge
front end	Kasutajapoolne liides
GAP	*Generic Access Profile* – üldine ligipääsu profiil
GATT	*Generic Attribute Profile* – üldine atribuutide profiil
GATT *client*	GATT klient ehk üle BLE ühenduv seade
GATT *server*	GATT server ehk seade, millega üle BLE ühendutakse
JSON	*JavaScript Object Notation* – süntaks andmete salvestamiseks ja vahetamiseks [3]
JVM	*Java Virtual Machine* – Java virtuaalmasin
kasutajaliides	Vahend, mis võimaldab kasutaja ja arvutisüsteemi vahelist suhtlust [4]
multiplatvorm	Erinevaid operatsioonisüsteeme/platvorme toetav
observer	Bluetooth GAP vaatleja roll
peripheral	Bluetooth GAP väline roll
rakendus	Lõppkasutajale loodud programm või programmide kogu [5]
string	Sõne, tervikuna käsitletav elemendi- või märgijada
timeout	Teadlik poolelioleva ülesande lõpetamine kindla aja möödudes, kui digitaalsel suhtlusel oodatavat vastust ei saada [6]

toiteplokk	Elektriseade, mida kasutatakse sisendvoolu tugevuse ja pinge toidetavale seadmele sobivaks muundamiseks [7]
Wifi	Juhtmevaba interneti standardi nimetus [8], võib kirjutada ka wi-fi. Kuna wi-fi saraneb *high fidelity* (kõrge täpsus) lühendile hi-fi, võib jääda mulje, et wi-fi on lühend terminist *wireless fidelity* (juhtmevaba täpsus), kuid wi-fi loojate sõnul see tõele ei vasta [9]
Wifi hotspot	Ala, kus on ligipääs juhtmevabale võrgule [10]

Sisukord

Jooniste loetelu

1 Sissejuhatus

Laboris olevaid toiteplokke kasutatakse näiteks elektroonika testimiseks, et kontrollida seadmete töötamist enne lõplikku süsteemi paigaldamist. Programmeeritavate ja logimisvõimekusega toiteplokkidega saab analüüsida koormuse muutumist erinevate tööparameetrite/sisendite korral. Samas logimisvõimekusega laboratoorsed toiteplokid, nagu näiteks Keysight E36300, prioritiseerivad väljundvõimsuse asemel väljundpinge ja -voolu täpsust. Seeläbi on eelnimetatud kolme väljundiga toiteploki (Keysight E36300) maksimaalne vool igal kanalil vaid 1 A.

Antud töös lahendab autor probleemi, et võimaldada juhitavus kuni kuue programmeeritava väljundiga toiteplokile. Autor võrdleb olemasolevaid toiteplokkide kasutajaliideseid ja viib koostöös toiteploki inseneriga läbi disainiprotsessi, mille käigus luuakse planeeritavale programmeeritavale toiteplokile kasutajaliides. Kasutajaliidese ja toiteploki planeerimine-disain on kooskõlastatud ning toimub samaaegselt. Antud toiteploki väljundpinge vahemik on 1 V – 40 V ja üksikute väljundite maksimaalne voolutugevus on 15 A, kokku kuni 90 A.

Kasutajaliides luuakse mobiilsele Android-seadmele ja valminud rakendus võimaldab antud toiteplokki üle juhtmevabalt kontrollida. Kasutajale kuvatakse erinevaid hetkenäitajaid ja logide põhjal genereeritud graafikuid. Lisaks võimaldab kasutajaliides toiteploki tööd graafiliselt programmeerida, näiteks väljundpinge ja voolutugevuse piirangu muutumist ajas. Sisseehitatud ekraani asemel saab protsessi kontrollida ja jälgida eemalt. Laboratoorsete toiteplokkidega võrreldes odavam hind suurendab kättesaadavust.

Antud lahendus sobib pigem suuremate süsteemide testimiseks kui madalama pingega töötavale väikeelektroonikale. Kuna kasutada saab kuni 6 kanalit korraga, saab ühe seadme erinevate pingetasemete tarbimist samaaegselt logida ja üksteise suhtes analüüsida. Võimalik on mõõta koormusgraafikuid. Kasutaja saab graafiliselt määrata, kuidas pinge kõver ajas muutuma hakkab. Muuhulgas on võimalik laadida erinevaid akuelemente ja akupakke ning analüüsida nende laadimiskarakteristikuid.

2 Probleemipüstitus

WatElectrical.com on defineerinud toiteplokki kui elektriseadet, mida kasutatakse sisendvoolu tugevuse ja pinge toidetavale seadmele sobivaks muundamiseks. Osad toiteplokid on eraldiseisvad, teised on oma kontrollitavatesse seadmetesse sisse ehitatud [7].

Antud bakalaureusetöö lahendatav probleem on, et on vaja juhtida konkreetset STM32WB55 mikrokontrolleri põhist iseseisvat toiteplokki ja kuvada erinevaid hetkeparameetreid. Selle toiteplokki põhiline kasutusala on päikesepaneelide regulaatorite ja erinevatel režiimidel akude testimine ning akude laadimine. Tema maksimaalne väljundvoolutugevus on kuni 90 A, väljundpingevahemik 1 V – 40 V. Samuti on tal olemas logimisvõimalus. STM32WB55 mikrokontrollerile on sisse ehitatud Bluetooth® Low Energy SIG specification v5.0 moodul (IEEE 802.15.4-2011 standard) [11].

Probleemi lahendamiseks on valitud luua Android-seadmel töötav juhtmevaba esialgu üle Bluetooth Low Energy töötav rakendus. Rakendus võiks kuvada hetkekarakteristikuid, võimaldada reguleerida toiteploki väljundeid ning kuvada logide põhjal informatsiooni kasutajale arusaadaval kujul.

2.1 Olemasolevate toiteplokkide kasutajaliideste võrdlus

Selles alapeatükis on autor võrrelnud erinevate toiteplokkide kasutajaliideseid ja nende funktsionaalsust. Kõikide kasutajaliideste kirjeldused pärinevad ametlikest allikatest, mille põhjal autor on teinud kokkuvõtted.

- AKTAKOM Power Manager (APM) – Windowsi rakendus, mis võimaldab reaalajas jälgida erinevate AKTAKOM toiteplokkide väljundeid, sisendeid graafiliselt ja tabeli abil programmeerida. Rakendus pakub ka võimalust väljundinfot ajavahemiku jooksul salvestada ning tagantjärele vaadata [12].

- AKTAKOM Smart Data Logger – Androidi rakendus, mis võimaldab salvestada AKTAKOM toiteplokkide andmeid logidesse ja vaadata graafikuid. Toiteplokiga

saab ühenduda kasutades Bluetoothi või RS-232 kaablit, mis ühendab toiteploki mobiilse seadmega läbi USB [13].

- VoltBot – Laadija ja toiteplokk LCD ekraani ja telefonirakendusega Androidile ja iOSile. Rakenduse kasutamiseks tuleb Android- või iOS-seade ühendada otse VoltBoti enda *wifi hotspot*iga. VoltBot lubab neljal eri kanalil kuni 4 A voolutugevust kasutada, pingel (2.50 – 12.50) V. Rakendus võimaldab kuvada graafikuid, mis kirjeldavad voolutugevuse, pinge ja võimsuse sõltuvust aja suhtes ja laadida alla Exceli formaadis algandmeid [14].

- Keysight E36300 seeria – Keysighti kolmekanalistel toiteplokkidel on sisseehitatud ekraan, mis võimaldab kõigi kanalite pinget ja voolutugevust korraga jälgida. Logide põhjal on võimalik kuvada energiakulu aja jooksul [15].

Olemasolevatest kasutajaliidestest kõige sarnasem on VoltBot, seetõttu võrdleme teda antud töös loodava lahendusega lähemalt. Loodaval lahendusel on 4 kanali asemel võimalik kasutada 6, pingevahemik (2.50 – 12.50) V asemel (1 – 40) V, voolutugevus ühel kanalil 4 A asemel 15 A, samuti on võimalik alustada tööd kanalitel eelseadistatud profiilidel korraga. Erinevalt VoltBotist on autori lahendusel võimalik toiteploki karakteristikuid graafiliselt programmeerida (näiteks voolutugevuse muutumine sõltuvalt ajast). Autori loodaval lahendusel esialgu aku laadimise jaoks eraldi profiilitüüpi ei ole ning keskendub ühele operatsioonisüsteemile multiplatvormi asemel. Mõlemad lahendused kuvavad statistika jaoks graafikuid.

2.2 Loodava rakenduse skoop

Autori eesmärk on luua toiteplokile eemalt juhitav kasutajaliides, mis suudab ka logidelt infot kuvada. Loodav rakendus keskendub *front end* poolele projektist, ehk sellele osale, millega kasutaja otseselt suhtleb. Arendatav kasutajaliides võimaldab kasutajal, kellel on mobiilne Android-seade, toiteplokiga juhtmevabalt ühenduda ja suhelda. Toiteplokk saadab loodavale kasutajaliidesele mõõdetavad andmed, mille kasutajaliides kasutajale kuvab. Kasutajaliides võimaldab kuvada ja toiteplokis käivitada erinevaid programme profiilide põhjal, mis salvestatakse toiteploki mällu. Toiteploki mälus hoitakse muuhulgas logisid, mille põhjal loodav loodes graafikuid ja muud infot kuvab.

Kuna toiteplokk ise pole kasutajaliidese loomisprotsessi alguses valmis, toimub arendus agiilselt (agiilne arendus), alustades teadaolevast. Esialgu on selleks vaated, mida kasutajale kuvatakse, täidetavad vormid ja saab alustada esmaste graafikute joonistamisega. Täpsem suhtlus toiteplokiga ja andmevahetusele kuuluvate *string*ide kuju selgub töö käigus.

Skoobist jääb välja akulaadimise funktsionaalsus, sealhulgas laadimise profiilitüüp ja aku mahtuvuse hindamine. Arenduse põhifookus on toiteplokist tulevate andmete kuvamisel ja talle käskude saatmisel. Arenduse käigus arvestatakse, et tulevikus võib suhtlus toiteplokiga toimuda lisaks Bluetooth Low Energyle ka üle serveri ja seetõttu peab rakendus olema antud suunas laiendatav.

3 Analüüs

Arenduse lihtsustamiseks analüüsis autor enne rakenduse ülesehitust ja protsesse, mis koos rakenduse põhiliste nõuetega olid eelnevalt toiteploki inseneriga kooskõlastatud.

Kasutajaliidesel on järgmised vaated, millele on võimalik navigeerida kasutades põhimenüüd:

- Status – staatuse vaade, kus kuvatakse hetkeparameetreid kõigi kanalite kohta, võimalus kanali töö lõpetada.
- Profiles – profiilide halduse vaade, kus saab luua uusi profiile ja neid sobivatesse gruppidesse salvestada, samuti vaadata olemasolevaid profiiligruppe koos profiilidega.
- Settings – rakenduse seaded, võimalus kanaleid üheks liita.
- Launch job – töö alustamise vaade, kust saab valida soovitud kanalitele profiilid ja käivitada programm ühel kuni kuuel kanalil korraga.
- Statistics – statistika vaade, kus saab logide põhjal näha statistikat ja kuvada graafikuid.

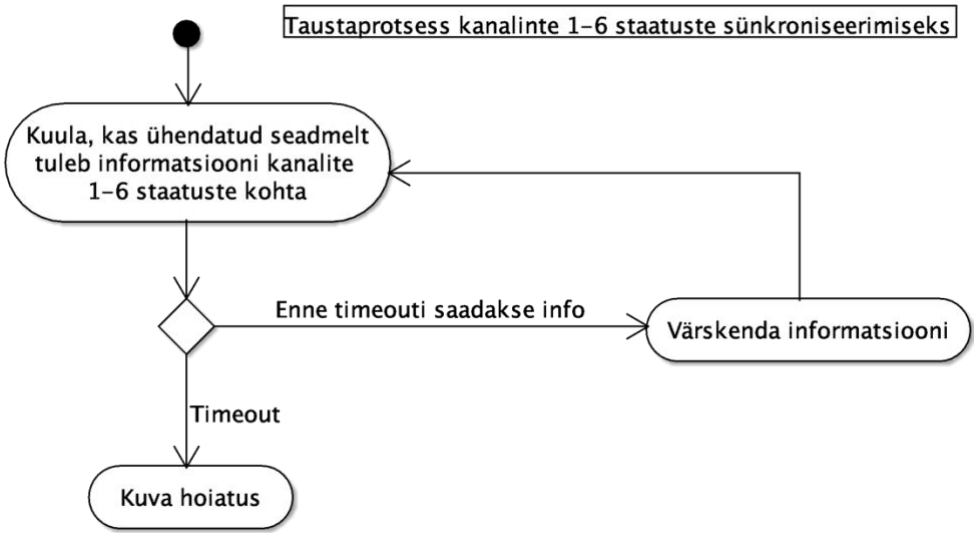
Järgnevalt on autor koostanud tegevusdiagrammid rakenduse eri protsessidele. Seadmete vahelise ühenduse loomisel on plaan kasutada standardset lahendust, seetõttu autor selle jaoks eraldi tegevusdiagrammi ei teinud.

Tegevusdiagrammid on koostatud järgmistele protsessidele:

- taustaprotsess kanalite 1–6 staatuste sünkroniseerimiseks,
- andmete pärimise protsess,
- kanali peatamise protsess,

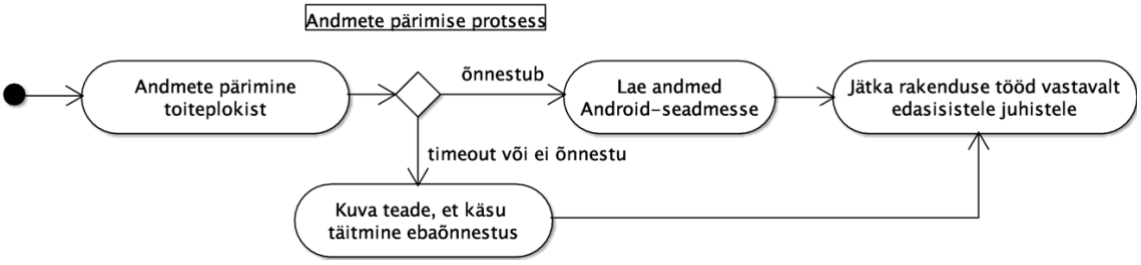
- profiili salvestamise ja laadimise protsessid,
- töö alustamise protsess.

Joonisel 1 on kirjeldatud rakenduse põhilist taustaprotsessi, milleks on pidev kuulamine, kas toiteplokk saadab kanalite staatuste kohta informatsiooni. Seni kuni toiteplokk infot saadab, eeldatakse, et ühendus on aktiivne. Saadud info põhjal uuendatakse rakenduses kõigi kanalite kohta kuvatavat informatsiooni (hetkenäitajad ja aktiivsus). Juhul kui info enne *timeout*i kohale ei jõua lõpetatakse kanalite staatuse uuendamine ja kuvatakse hoiatus, et ühendusega võib probleem olla.



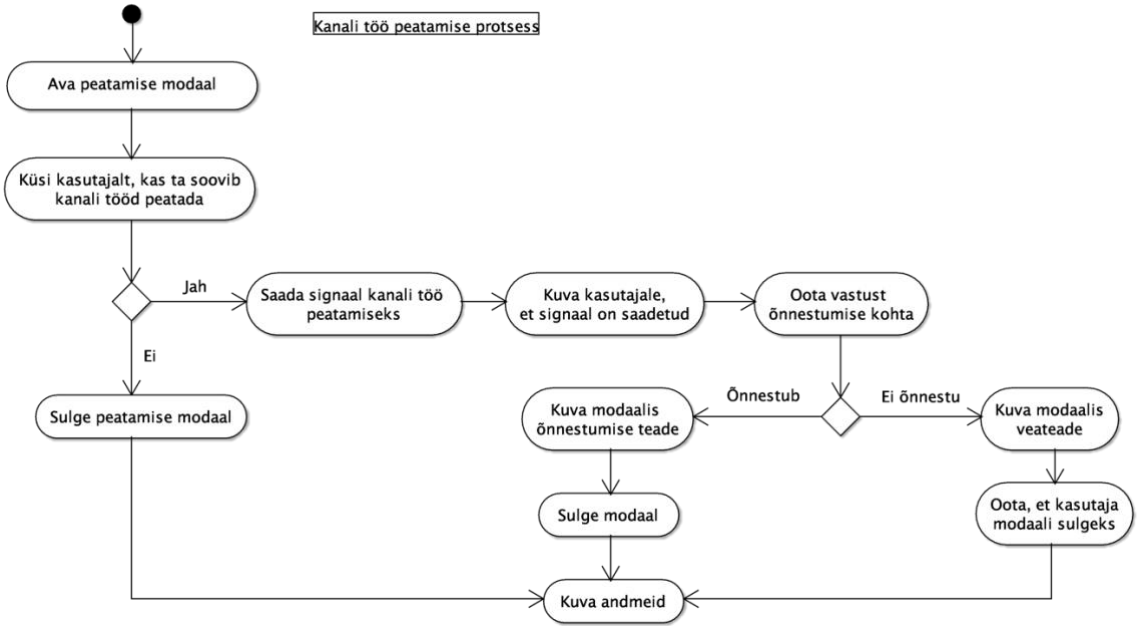
Joonis 1. Taustaprotsessi tegevusdiagramm.

Joonisel 2 on kirjeldatud andmete pärimise protsessi. Seda kasutatakse näiteks profiilide, profiiligruppide ja logide pärimiseks, kuid ka toiteploki tööprotsessi käivitamiseks või peatamiseks. Toiteplokile saadetakse päring, millele oodatakse vastust. Kui päring õnnestub ja toiteplokk tagastab sobival kujul info, siis uuendatakse Android-seadmes vastavad muutujad. Edasised tegevused sõltuvad sellest, mis olukorras andmeid päriti.



Joonis 2. Andmete pärimise protsessi tegevusdiagramm.

Joonisel 3 on kirjeldatud valitud kanali töö peatamise protsessi. Seda saab esile kutsuda Status vaatest, kus on kuvatud kanalite 1 – 6 hetkeseisud. Peatada saab ainult aktiivseid kanaleid, mis on parajasti töös. Kui kasutaja valib aktiivse kanali, mille tööd ta peatada soovib, küsitakse talt enne, kas ta on kindel oma soovis kanali töö peatada. Selleks avatakse modaal, et kasutajat protsessiga käiguga kursis hoida. Juhul kui kasutaja kanali peatamisest loobub suletakse modaal ja kasutajale kuvatakse taas kanalite staatused. Kui kasutaja soovib peatamisega jätkata, saadetakse signaal valitud kanali töö peatamiseks toiteplokile ja oodatakse vastust. Kasutajale kuvatakse kuni vastuse saamise või *timeout*ini toimuva kohta infot. Kui kanali töö peatamine õnnestub sulgetakse modaal automaatselt, vastasel juhul jääb kasutajale võimalus veateatega tutvuda kuni ta modaali ise sulgeb. Peale modaali sulgemist jätkatakse Status vaates andmete värskendamist ja kuvamist.

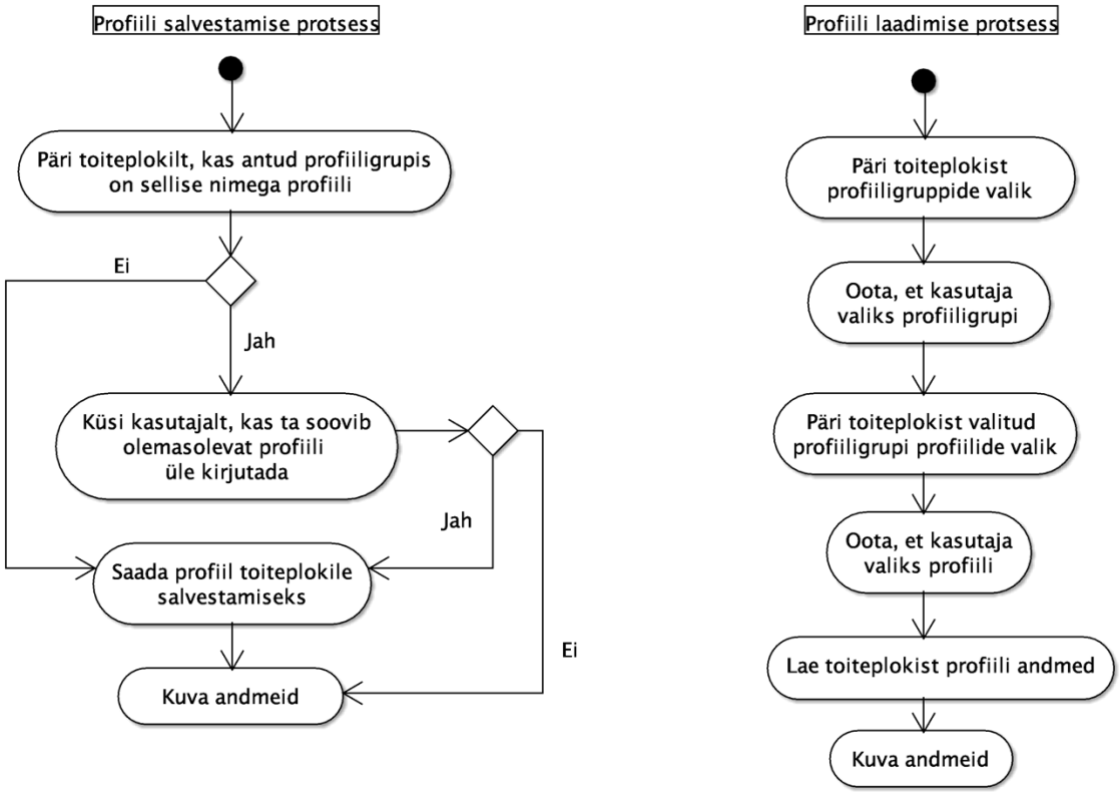


Joonis 3. Kanali töö peatamise protsessi tegevusdiagramm.

Joonisel 4 on kirjeldatud profiili salvestamise ja laadimise protsessid. Siinkohal toimub toiteplokiga suhtlemine ja andmete pärimine nagu kirjeldatud joonisel 2.

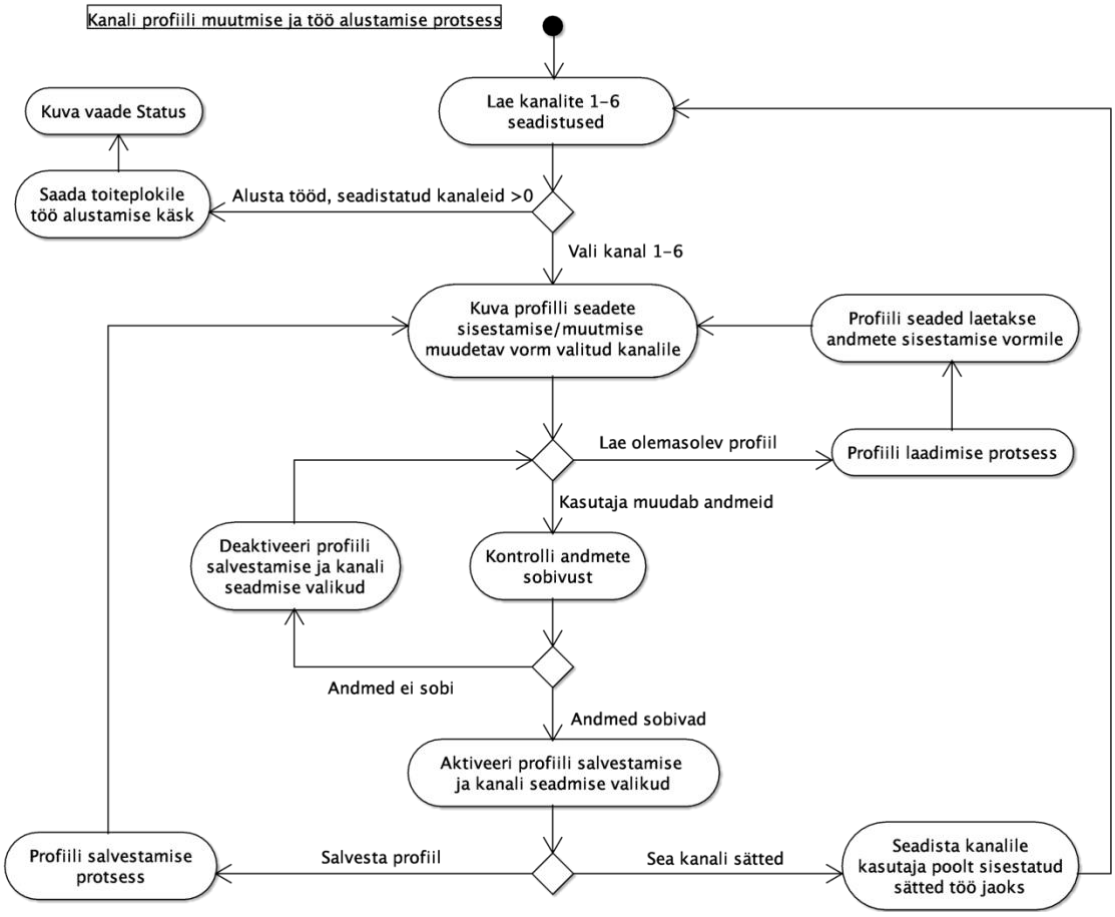
- Profiili salvestamisel luuakse kas uus profiil või kirjutatakse eelnev üle. Seetõttu päritakse toiteplokilt, kas sellise nimega profiil juba eksisteerib profiiligrupis kuhu teda salvestada üritatakse. Sel juhul küsitakse kasutajalt, kas ta soovib eksisteerivat profiili üle kirjutada. Kui profiili ei eksisteeri või kasutaja soovib eksisteerivat üle kirjutada jätkatakse profiili salvestamisega.

- Profiili laadimisel päritakse toiteplokist esimesena profiiligrupid, mille hulgast kasutaja peab valiku tegema. Alles seejärel päritakse profiilid. Kui kasutaja on profiili välja valinud, laetakse vastavad andmed toiteplokist Android-seadmesse. See, mida andmetega edasi tehakse, sõltub vaatest, kuhu profiili laeti (profiili loomine/muutmine või töö alustamine).



Joonis 4. Profiili salvestamise ja laadimise protsessid.

Joonisel 5 on kirjeldatud kanali profiili muutmise ja töö alustamise protsessi, mis käivitub kui avada töö alustamise vaade (Launch job). Esialgu laetakse kanalite 1 – 6 seadistused, kuvatakse ainult mitteaktiivsed, mis hetkel pole töös. Alguses on kanalid seadistamata ja kasutaja peab valima, millist kanalit ta seadistada soovib. Konkreetse kanali valimise järgselt on kasutajal võimalik seadeid endale sobivaks kohandada, sealhulgas laadida seaded olemasolevalt profiililt (profiili laadimise protsess joonisel 4). Kui seaded vastavad nõuetele tekib valik profiil salvestada (profiili salvestamise protsess joonisel 4) või seada kanalile. Viimasel juhul suunatakse kasutaja tagasi võimalike kanalite seadistamise koondvaatele, kust nüüd on võimalik tööd alustada või jätkata kanalite seadistamist. Kui soovitakse tööd alustada, saadetakse toiteplokile vastav käsk ja seejärel suunatakse kasutaja vaatele Status, kust ta saab näha kõikide kanalite jooksvat hetkeseisu.



Joonis 5. Kanali profiili muutmise ja töö alustamise protsessi tegevusdiagramm.

Joonisel 5 kirjeldatud töö alustamise protsess sisaldab lisafunktsionaalsus Status vaatest üksiku kanali töö alustamisega võrreldes. Launch job võimaldab tööd alustada korraga mitmel kanalil ja soovi korral lisada igale kanalile viitaeg.

3.1 Androidi toetava sobivaima tehnoloogia valik

3.1.1 Keelevalik

Tehnoloogia valikul on põhirõhk võimalikult hea ühilduvus Android-platvormiga, teiste seadete toetus on vähem oluline. Kasutav tehnoloogia peab efektiivselt suutma kasutada Bluetoothi ja võimaldama graafikute joonistamist. Mõnevõrra oluline on ka õpitavuskeerukus autorile, kuna see mõjutab ajakulu ning tõenäoliselt ka rakenduse kvaliteeti.

Java ja Kotlin on Androidi looja Google'i poolt ametlikult toetatud ning jooksevad Androidi JVM (*Java Virtual Machine*) peal [16]. Mõlemat on võimalik kasutada

arenduskeskkonnas Android Studio, mis toetab arendust näiteks visuaalse disainimise tööriista ja soovituste abil [17]. Kotlini põhiliseks miinuseks tuuakse võrdlevates allikates seda, et sellel pole niivõrd suurt infotuge ja palju teeke saadaval kui Javal [16], [17].

Samas kui arenduse käigus peaks tekkima vajadus kasutada Java-põhist teeki, siis on võimalik Kotlinit ja Javat kõrvuti samaaegselt kasutada [17]. Kuigi madalatasemeline C/C++ võimaldaks ehk parimat jõudlust ja otsest suhtlust riistvaraga, on temaga programmeerimine Javast keerukam ja võimaldab vigade kergemat tekkimist [16], [17], samuti puudub autoril nende keeltega kokkupuude.

Veel oleks variant kasutada Xamarini (C# keel), kuid võrreldes Androidi süsteemi otseselt toetatud keeltega, nagu Java, toob see kaasa rakenduse suurenemise ja võimalikud stabiilsuse probleemid [18]. Seetõttu pole Xamarin esmane valik, kuigi autoril C# keelega on kogemus olemas.

JavaScripti raamistike, nagu näiteks React Native, eeliseks võimalus kirjutada üks rakendus, mis töötab korraga erinevatel süsteemidel nagu Android, iOS ja UWP; seevastu kipuvad JavaScripti raamistikud teiste lahendustega võrreldes ebaefektiivsemalt töötama ning kuigi React Native jaoks on mitmeid valmis komponente, tuleks võimalikud puuduolevad moodulid mõnes Androidiga otsesemalt ühilduvas keeles kirjutada [19].

Selles lõigus võtab autor kokku SPEC INDIA avaldanud artiklis [20] toodud Java ja Kotlini võrdlusest silma jäänud tähtsamad argumendid. Java ja Kotlin kompileeritakse lõpptulemusena samaks koodiks, kuid Kotlini eesmärk on Androidi rakenduse koodikirjutamist optimeerida. Olles küllaltki uus keel, pole Kotlinil nii suur kasutajaskond kui Javal, kuid tema populaarsus on kiiresti kasvanud. Kotlin lubab olla täpne, efektiivne, kiirem töövahend kui Java ja vähendada koodi kirjutamisel tekkivate vigade hulka. Lisaks pole Kotlini õppimine Java taustaga inimesele keeruline ning keel sobib kasutamiseks olemasolevate Java teekidega.

Kuigi C/C++ keeled võimaldaks oskusliku kasutamise läbi ehk luua kõige optimaalsem rakendus, on need autorile võõrad ja nendega arendamine aeganõudev. Analüüsis toodud põhjustel, eelistab autor kasutada Android Studio keskkonda ja arenduskeelena kas Javat või Kotlini. Autori valikuks jääb Kotlin, kuna Kotlini koodistiil tundub puhas, aitab ära hoida võimalikke vigu, on Java taustalt tulles võrdlemisi lihtsasti omandatav ning vajadusel saab teda ka Javaga kombineerida.

3.1.2 Ülevaade BLE-st ja sellest tulenevad nõuded/piirangud

STM32WB55 mikrokontrolleril on sisse ehitatud BLE (*Bluetooth Low Energy*) [11]. BLE sai valitud toiteploki ja Android-seadme vahelise esialgse suhtluse teostamiseks ning rakenduse arendamisel on vaja BLE toega arvestada.

Bluetoothi ametlikus juhendis arendajatele [21] on lähemalt räägitud BLE profiilidest. *Genereic Attribute Profile* (GATT) võimaldab defineerida tabeli, milles kirjeldatakse seadme hetkestaatust ja võimalikke operatsioone. Seade, mis teenust, karakteristikuid ja tunnuseid majutab, on GATT *server* ja seade, mis tema poole pöördub on GATT *client*. *Generic Access Profile* (GAP) vastutab selle eest, kuidas ühenduse loomine toimub. Juhendis on kirjeldatud nelja alljärgnevat profiili.

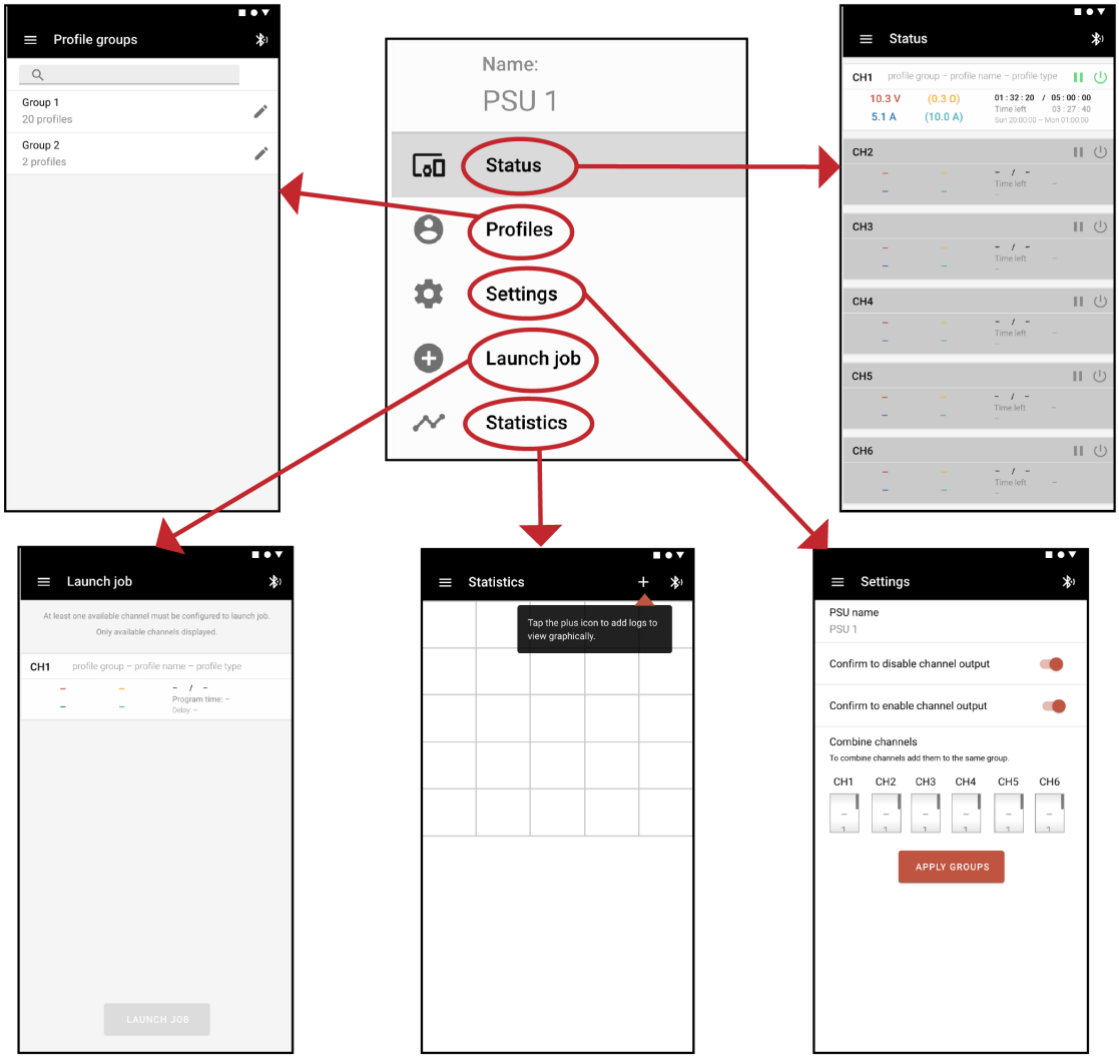
- *Peripheral* – reklaamib, kutsub ja aktsepteerib ühendusi *central* rolliga seadmetelt.
- *Central* – skännib ja otsib reklaampakette, võib otsustada ühenduda sobiva seadmega.
- *Broadcaster* – reklaamib, kuid ei aktsepteeri ühendusi.
- *Observer* – skännib ja töötleb reklaampakette, kuid ei proovi kunagi ühendust luua.

Eelmises peatükis kirjeldatud definitsioonide kohaselt on Android-seade GATT *client*, mis ühendub GATT *server*ina töötava toiteplokiga. Toiteploki roll on *peripheral*, kuna tema ülesanne on ühenduse loomisel ennast välja reklaamida. Android-seade rolliga *central* peaks suutma toiteploki tema reklaampakettide abil üles leida ja avaldama soovi ühenduda, mille toiteplokk seejärel aktsepteerib.

Esimesed BLE toega Androidi versioonid (4.3 ja 4.4) olid probleemsed nii ühenduse loomise kui rakenduse töös hoidmise poolest; Android 5.0 pakub arendajale paremaid skaneerimise võimalusi ja on üldiselt stabiilsem [22]. Seetõttu on ka autor otsustanud valida Androidi versiooniks 5.0. Et tagada rakenduse võimalikult lai seadete tugi, sellest kõrgemat versiooni ei kasutata, eeldusel, et antud versioonil on arendamiseks vajalik funktsionaalsus olemas.

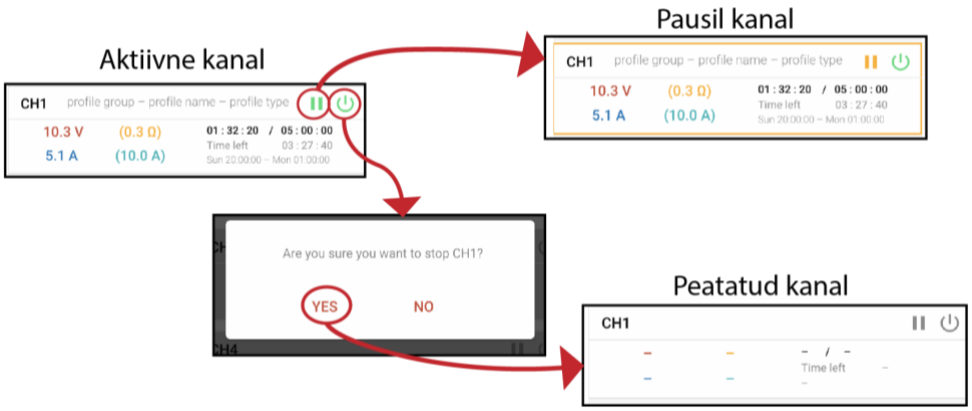
3.2 Kasutajaliidese disain

Kasutajaliidese disaini jaoks kasutas autor UXPin veebiplatvormi graafilise prototüübi loomiseks [23]. Prototüübi mõõtmed on loodud Androidi telefoni Google Pixel 2 põhjal, mis oli UXPinil endal valmis. Antud seade ühildub mitmete kasutusel olevate ekraanide mõõtmetega, sealhulgas autorile kättesaadavate seadetega. Prototüübi lõi autor, et see oleks abiks rakenduse läbi mõtlemisel, täpsema kooskõlastamise jaoks toiteploki inseneriga enne arendamist ning aluseks rakenduse kirjutamisel.



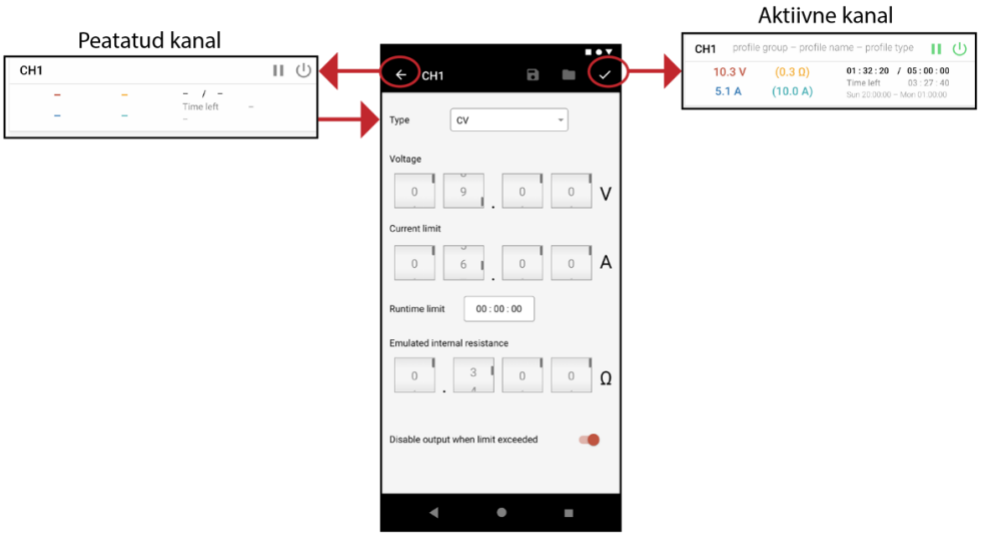
Joonis 6. Rakenduse prototüübi navigeerimisvõimalused põhimenüüst.

Rakenduse üleval osas on musta värvi tööriistariba. Must värv sulandub enamike seadmete korpusega ja tõmbab vähem tähelepanu, võimaldades keskenduda paremini rakenduse põhiosale. Tööriistariba vasakus servas olev kolme triibuga ikoon võimaldab avada põhimenüü koos navigeerimisvalikutega Status, Profiles, Settings, Launch job ja Statistics, mis viivad vastavatele vaadetele (joonis 6).



Joonis 7. Aktiivse kanali pausile panek ja peatamine.

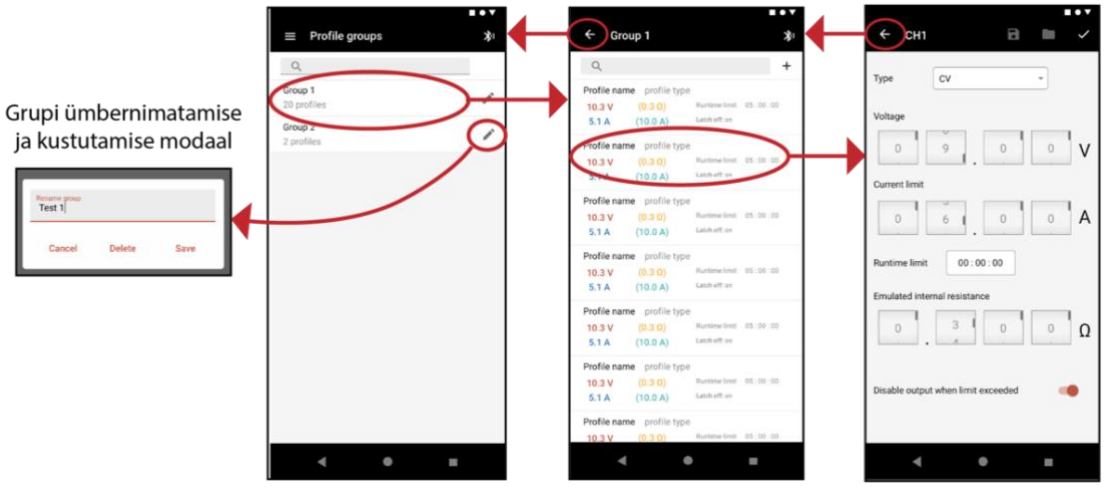
Vaates Status, kuvatakse toiteploki kanalite hetkestaatuseid. Aktiivne kanal näitab hetkel käimasoleva profiili andmeid (grupi nimi, profiili nimi, profiili tüüp) ja karakteristikuid (nt pinge, voolutugevus). Aktiivset kanalit on võimalik pausile panna või tema töö peatada. Peale kanali töö peatamist on võimalik valida kanalile uus profiil. Prototüübis on võimalik seda voogu proovida vaid esimese kanali näitel. Profiili seadmise vaatest on võimalik liikuda tagasi, mille tulemusel peatatud kanalile uut profiili ei lisata; vastasel juhul, kui andmetega ollakse rahul, on võimalik käivitada kanali töö, millisel juhul suunatakse kasutaja automaatselt tagasi vaatesse Status (joonis 8).



Joonis 8. Peatatud kanalilt on võimalik liikuda vaatesse, kust saab alustada profiilipõhist tööd.

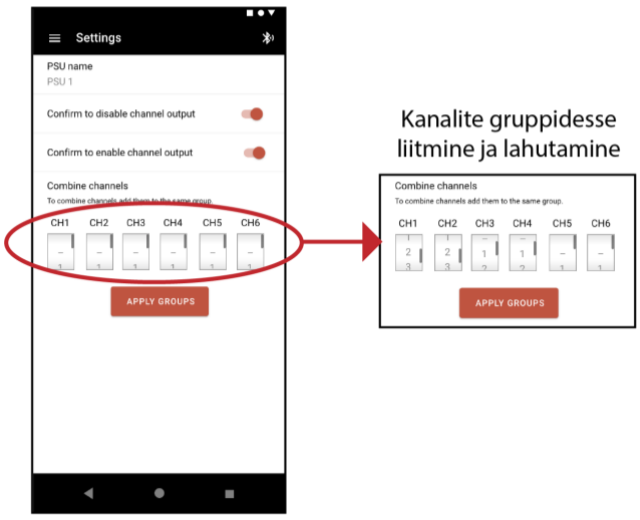
Joonis 9 kirjeldab profiili redigeerimise voogu. Kõik profiilid kuuluvad gruppidesse. Profiiligruppide loetelus on igal profiiligrupil redigeerimise nupp, mis võimaldab muuta profiiligrupi nime või kustutada terve grupp koos temas sisalduvate profiilidega. Profiiligrupi avades, vajutades selleks alale, kus on kirjeldatud tema andmed, saab näha

temas sisalduvaid profiile koos andmetega. Profiili saab redigeerida avades temale vajutades tema redigeerimise vaade.



Joonis 9. Profiiligrupi ümbernimetamine/kustutamine ja profiili redigeerimise voog.

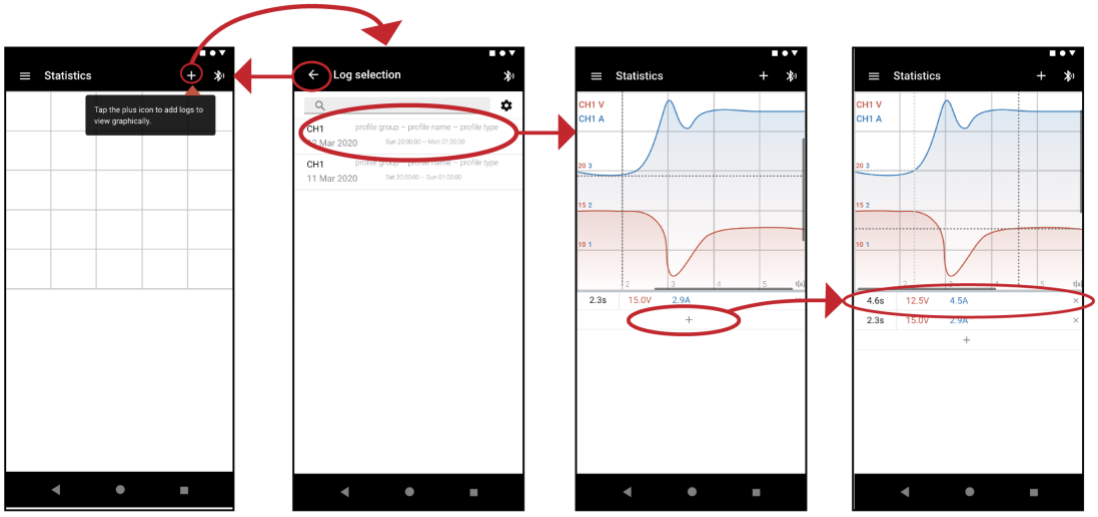
Joonisel 10 on toodud näiteid võimalikest seadetest, mis mõjutavad nii rakenduse kui ka toiteploki enda käitumist. Kasutaja saab valida, kas soovib hoiatusteateid enne kanali töö alustamist või peatamist. Toiteplokil on võimalik ära muuta tema nimi ning liita/lahutada kanaleid. Viimane töötab põhimõttel, et kanalid, millel on sama grupeerimisnumber liidetakse üheks, seega on kuue kanali puhul võimalik luua kuni kolm gruppi. Grupeerimise jõustumiseks on vaja vajutada vastavat nuppu.



Joonis 10. Settings vaade kanalite gruppidesse liitmise ja lahutamisega.

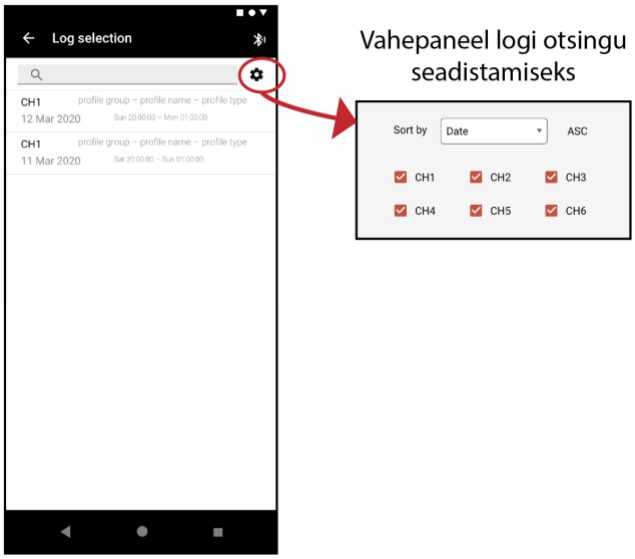
Vaates Statistics on võimalik vaadata ja võrrelda logisid graafiku abil. Andmeid saab korraga vaadata kuni kahe logi kohta. Joonisel 11 on näidatud kuidas lisada kuvamiseks ühte logi ja valimise järgselt lisada võrdlustabelisse punkte. Punkt tuleb graafikul puute

abil enne ära fikseerida ja seejärel saab ta alumise pluss ikooni vajutamise abil võrdlustabelisse lisada. Prototüübist on puudu Statistics vaate seaded.



Joonis 11. Statistics vaatesse logide valimine ja graafikule punktide lisamine.

Logide leidmist lihtsustab otsing, mis otsib teksti üle profiili andmete (nimi, grupi nimi, profiilitüüp). Vajutades ikooni otsinguriba kõrval, saab otsingutulemusi sorteerida näiteks kuupäeva järgi kasvavalt/kahanevalt ning filtreerida kanalite järgi (joonis 12).



Joonis 12. Statistics vaate logide otsing ja tema seaded.

Erinevalt siin kirjeldatud disaini joonistest on prototüüp (lisa 1) mingil määral interaktiivne ja sisaldab muuhulgas animatsioone. Tõenäoliselt teeb autor arenduse käigus disainis muudatusi – näiteks saadud tagasiside põhjal, et tekst võiks olla suurem.

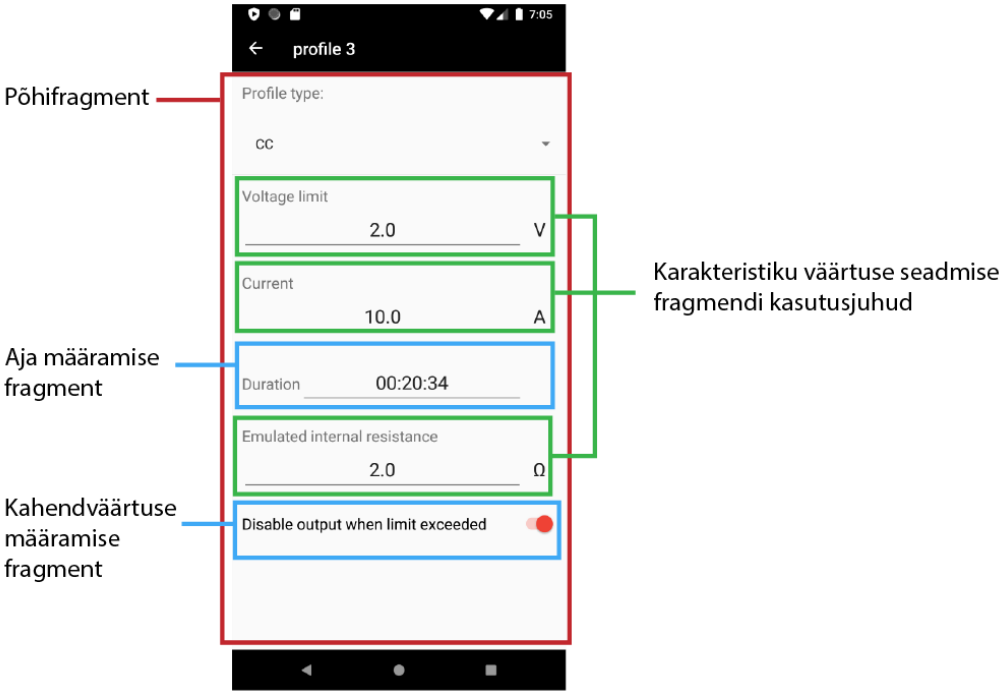
4 Rakenduse loomine

Arhitektuuri loomisel püüdis autor lähtuda heast tavast hoida lahus vaated, äriloogika ja andmetüübid, et rakendus oleks tulevikus kergemini laiendatav. Arhitektuuri skeem on leitav lisast 2. Kuna toiteplokki ennast veel polnud, seadis see piirangud andmevahetuse testimisele, kuid kuna algne disain oli paigas, siis alustas autor vaadete ja navigatsiooni loomisest. Kuigi põhimenüü arvestab kõigile plaanitavatele vaadetele pääsemisega, keskendus autor esialgu kahele: Status ja Profiles. Lisaks on antud bakalaureuse töö raames loodud eelnimetatud vaadete alamvaated, nende kuvamise loogika sõltuvalt andmetest ning andmete teisendamine kuvamise-muutmise jaoks sobivale kujule. Arenduse käigus on autor arvestanud sellega, et alamvaadetel tehtud andmemuudatused peavad olema edaspidi kättesaadavad. Rakenduse kood on kättesaadav GitHubis [24].

4.1 Navigatsioon ja vaadete loogika

Rakenduse navigatsiooni aluseks võttis autor Android Studio poolt pakutavad mallid, kombineerides ühest mallist avatava menüü (Navigation Drawer Activity) loogika ja teistest noolekese abil tagasi navigeerimise loogika (Basic Activity). Avatava menüü sisu kirjeldati juba joonisel 6 ning siin alapeatükis eraldi ei käsitleta. Selleks, et andmed seadme pööramise tulemusel kaotsi ei läheks ja vaade kuvamisega tekkivaid potentsiaalseid probleeme vähendada, kuvatakse rakendust vaid vertikaalselt. Juhul kui kogu informatsioon vertikaalselt ekraanile korraga ära ei mahu, muutub vaade keritavaks. See võimaldab ka rakenduse paremat toimetulekut informatsiooni mahutamisel väiksematele ekraanidele.

Ekraanil kuvatav tervikvaade on kokku pandud väiksematest osadest ehk fragmentidest, kusjuures võivad eri fragmendid sisaldada omakorda alamfragmente. Samasuguseid vaateid korduvkasutatakse. Näiteks profiilide fragmendi kuvamisel määratakse kuvatavad alamfragmendid programmeerimise teel, mille abil saadetakse talle põhifragmendist täpsemad detailid.



Joonis 13. Fragmentide kasutus profiili redigeerimisel.

Joonisel 13 on kirjeldatud alamfragmentide kasutust põhifragmendis profiili redigeerimise vaate näitel. Antud põhifragment sisaldab profiilitüübi rippmenüüd pealkirjaga, valitud on profiilitüüp CC (*constant current* ehk püsiv voolutugevus). Antud profiilitüübi puhul liidetakse programmeerimise abil põhifragmenti järgmised alamfragmendid:

- kolm kasutusjuhtu karakteristiku väärtuse seadmise fragmendist (pinge piirang, voolutugevus, emuleeritud sisetakistus),
- üks aja määramise fragment (kestus),
- üks kahendväärtuse määramise fragment (kas lülitada väljund välja, juhul kui piirang ületatakse).

Fragmentide lisamise koodi loetavuse parandamiseks tegi autor abiklassi FragmentManagerHelper, mis sisaldab meetodeid eri liiki fragmentide lisamiseks (kasutus näidatud joonisel 14). Need meetodid võimaldavad luua alamfragmentidest kasutusjuhud, andes igale juhule kaasa teda eristavad parameetrid. Antud juhtudel antakse meetoditesse kaasa muudetava väärtuse pealkiri, kuvatav algväärtus ja karakteristikute puhul ühik.

```
private fun addFragmentsForProfile(profile: ProfileDTO) {
    val fragmentManagerHelper =
        FragmentManagerHelper(
            childFragmentManager,
            R.id.linearLayout_fragment_profile_type_container
        )

    fragmentManagerHelper.addSetParameterFragment("Voltage
limit", profile.voltageLimit, "V")
    fragmentManagerHelper.addSetParameterFragment("Current",
profile.current, "A")
    fragmentManagerHelper.addSetTimePropertyFragment("Duration",
profile.duration)
    fragmentManagerHelper.addSetParameterFragment(
        "Emulated internal resistance",
        profile.resistance,
        "Ω"
    )
    fragmentManagerHelper.addSetBooleanFragment(
        "Disable output when limit exceeded",
        profile.latchOff
    )
}
```

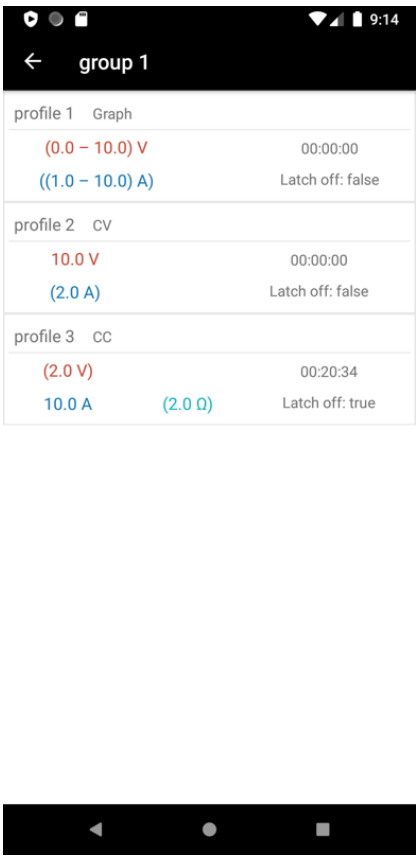
Joonis 14. Profiilitüübi alamfragmentide lisamise meetod addFragmentsForProfile CC profiili näitel.

FragmentManagerHelperi meetodite kirjeldamiseks toob autor näiteks addSetParameterFragment (joonis 15). Meetodile antakse kaasa vaatesse soovitavad väärtused – pealkiri, komakohaga arvväärtus ja ühik – ning luuakse nende põhjal uus fragment, mis lisatakse FragmentManagerHelper konstruktoris defineeritud vaatesse. Meetod tagastab loodud fragmendi Kotlini klassi, et tulevikus oleks võimalik ligi pääseda tema muutujatele ja need terviklikult koos muudatustega profiili salvestada.

```
fun addSetParameterFragment(title: String, value: Float?, unit:
String): SetParameterFragment {
    val parameterFragmentInstance =
        SetParameterFragment.newInstance(
            title,
            value ?: 0F,
            unit
        )
    fragmentManagerToUse.beginTransaction()
        .add(parentLayoutId, parameterFragmentInstance).commitNow()
    return parameterFragmentInstance
}
```

Joonis 15. Abimeetod addSetParameterFragment.

Sama tüüpi andmete üksteise all kuvamiseks pakub Android vaadet nimega RecyclerView, mis võimaldab üksiku andmeobjekti kuvamise plokki eraldiseisvalt disainida. Autor kasutas RecyclerView võimalust neljas kohas: kanalite staatuste kuvamisel, profiiligruppide ja profiilide loeteludes üksikandmete kuvamisel ning graafi profiili punktide lisamise/eemaldamise korduvkasutatavas fragmendis, kus igal punktidega kirjeldataval karakteristikul tekib sama koodi kasutades kaasa oma RecyclerView.



Joonis 16. RecyclerView kasutus grupis sisalduvate profiilide kuvamiseks.

Joonisel 16 kuvatakse RecyclerView abil näidisandmetega kõigi kolme antud bakalaureuse töö raames lisatud profiilitüübi näidisprofiile.

- CV – *constant voltage* ehk püsiv pinge, mis on määratletud konstantse pinge ja voolutugevuse piiranguga.
- CC – *constant current* ehk püsiv voolutugevus, mis on määratletud konstantse voolutugevuse ja pinge piiranguga.

- Graph – graafilise programmeerimise profiil, kus väärtused võivad tingimustest sõltuvalt muutuda.

Andmete kuvamiseks peab komponendi tekstiväljadele ütlema, mida kuvada. Seda aitavad teha enum-klassis nimega ProfileType olevad meetodid, mis võtavad arvesse profiilitüübi ja vormindavad andmed vastavalt antud profiili profiilitüübile ettenähtud kujule (joonis 17).

```
enum class ProfileType(val profileTypeName: String) {
    CV("CV") {
        override fun formatField2(profile: ProfileDTO) =
            Format.toLimit(profile.currentLimit, LimitType.C)
    },
    CC("CC") {
        override fun formatField1(profile: ProfileDTO) =
            Format.toLimit(profile.voltageLimit, LimitType.V)
    },
    Graph("Graph") {
        override fun formatField1(profile: ProfileDTO): String =
Format.toGraphVoltage(profile)
        override fun formatField2(profile: ProfileDTO): String =
Format.toGraphCurrent(profile)
    };

    open fun formatField1(profile: ProfileDTO): String =
Format.toVoltage(profile.voltage)
    open fun formatField2(profile: ProfileDTO): String =
Format.toCurrent(profile.current)
    open fun formatField3(profile: ProfileDTO): String = ""
    open fun formatField4(profile: ProfileDTO): String =
Format.toResistance(profile.resistance)
    open fun formatTimeField1(profile: ProfileDTO): String =
profile.duration.toString()
    open fun formatTimeField2(profile: ProfileDTO): String = "Latch
off: " + profile.latchOff
}
```

Joonis 17. Enum-klass ProfileType sisaldab teisendusmeetodeid, mis võimaldavad vastavalt profiilitüübile kuvada erinevat stringi.

Väljade puhul, mille vormindamine profiilist tulenevalt põhimeetodist erineb, kasutas autor ülekatet. Näiteks on ülekatet kasutatud joonisel 17 meetodi formatField1 puhul profiilitüüpidel CC ja Graph. Kõikidele profiilitüüpidele kutsutakse andmete vormindamiseks välja sama meetod, kuid kui profiilil, mille jaoks andmeid kuvatakse, on erisusi, muudetakse meetodi sisu ülekatet kasutades vastavalt ära. Tulemuseks on, et samal väljal on võimalik kuvada CV profiili puhul pinget, CC profiili puhul pinge

piirangut, mida tähistatakse sulgudega, ning Graph puhul pinge kõikumise vahemikku (joonis 16 punasega toodud väärtus). Tekstiväljade profiilist tulenevate teisendusmeetodite kasutus on kirjeldatud joonisel 18. Vastav profiilitüüp sisaldub välja vormindamise meetodisse kaasa antavas profiili objektis.

```
private fun setTextViews(profile: ProfileDTO) {
    val currentProfile = profile.profileType

    mProfileName?.text = profile.profileName
    mProfileType?.text = profile.profileType.profileTypeName
    mProfileField1?.text = currentProfile.formatField1(profile)
    mProfileField2?.text = currentProfile.formatField2(profile)
    mProfileField3?.text = currentProfile.formatField3(profile)
    mProfileField4?.text = currentProfile.formatField4(profile)
    mProfileTime1?.text = currentProfile.formatTimeField1(profile)
    mProfileTime2?.text = currentProfile.formatTimeField2(profile)
}
```

Joonis 18. Profiili tekstiväljade kuvatava teksti teisendamise meetodite kasutamine.

Staatuste ja profiiligruppide kuvamise protsess on sarnane profiilide omale, kuid sisaldab vähem eritingimusi. Profiilid tõi autor näiteks, kuna tema meelest oli profiilide nimekirja kuvamine ja profiili redigeerimise loogika originaalsem kui teiste vaadete puhul, kus standardist eristuvat keerukust oli vähem.

4.2 Andmed

Andmete vahetamiseks leppis autor toiteploki inseneriga kokku, et kasutatakse JSON-formaati (näide lisas 3), kuna selle tugi on olemas nii Kotlinis/Javas kui ka toiteploki C++ keele jaoks ning samuti on see formaat kergesti inimloetav. Olemasolevate läbiproovitud teekide kasutamine on kiirem ning vähem veaohtlik kui ise uue lahenduse välja mõtlemine. Kotlini poole pealt otsustas autor JSONi objektideks teisendamiseks kasutada välist ametlike Kotlini arendajate loodud kotlinx.serialization teeki [24], kuna on see on mugavam kui sisemiste teekide kasutamine ning, võrreldes teiste väliste teekidega, on autori meelest suurem tõenäosus, et tulevikus tema tugi Kotlinis jätkub.

```
@Serializable
data class Profiles (val profiles: Array<Profile>)

@Serializable
data class Profile(
    val name: String,
    val group: String,
    val type: String,
    val duration_ms: Long?,
    val value: Float? = null,
    val limitValue: Float? = null,
    val resistance: Float? = null,
    val latchOff: Boolean? = false,
    val graph: Graph? = null
)
```

Joonis 19. Koodinäited olemiklassidest Profiles ja Profile, mida autor kasutas JSONi vastava tüübiga objektideks teisendamiseks.

Selleks, et kotlinx.serialization töötaks, on vaja olemiklassi, annotatsiooniga @Serializable (joonis 19), kusjuures peab ära annoteerima ka kõik temas viidatud olemiklassid, antud juhul Profiles ja temas sisalduv Profile (ja temas omakorda sisalduv Graph jne). Paremaks ühildumiseks kasutajale kuvatavate vaadete ja andmete töötlemiseks, otsustas autor klassi muutujad eraldi olemiklassi põhjal ära kaardistada. Klassile Profile vastav klass ProfileDTO on välja toodud joonisel 20.

```
data class ProfileDTO(
    var profileName: String = "<template>",
    var profileGroupName: String = "<template>",
    var profileType: ProfileType = ProfileType.CV
) {
    var voltage: Float? = null
    var current: Float? = null
    var voltageLimit: Float? = null
    var currentLimit: Float? = null
    var resistance: Float? = null
    var duration: String? = null
    var latchOff: Boolean? = null
    var graph: GraphDTO? = null
}
```

Joonis 20. Profile klassile vastav rakendusele sobivaks teisendatud andmetega klass ProfileDTO.

Profile objekti ProfileDTO objektiks ümber kaardistamine on kirjeldatud joonisel 21. Põhiline erinevus Profile objekti ümberkaardistamisel tuleb profiilitüüpidega (profileType välja põhjal) andmete teisendamisest.

```
fun mapProfile(profile: Profile): ProfileDTO {
    val profileDTO = ProfileDTO(profile.name, profile.group,
ProfileType.valueOf(profile.type))
    profileDTO.resistance = profile.resistance
    profileDTO.latchOff = profile.latchOff
    profileDTO.duration = Format.millisToTime(profile.duration_ms)

    when (profileDTO.profileType) {
        ProfileType.CV -> {
            profileDTO.voltage = profile.value
            profileDTO.currentLimit = profile.limitValue
        }
        ProfileType.CC -> {
            profileDTO.current = profile.value
            profileDTO.voltageLimit = profile.limitValue
        }
        ProfileType.Graph -> {
            profileDTO.graph = mapGraph(profile.graph!!)
        }
    }
    return profileDTO
}
```

Joonis 21. Profile tüüpi objekti ümberkaardistamine ProfileDTO tüüpi objektiks.

Autorile tundus mugavam pigem Profile klassi väljad ühekordselt ümber teisendada, kui hakata andmete salvestamiseks keerukamat loogikat välja mõtlema. Erinevalt Profile klassist on ProfileDTOs iga väärtuse jaoks eraldi parameeter selle asemel, et ühe välja sisu tõlgendus sõltuks teise välja väärtusest, väärtus kas on olemas või ei ole.

4.3 Testimine

Testimiseks kasutas autor jooksvalt Android Studio poolt pakutavat emulaatorit ja logimist. Logimisvõimalus tuleb Android Studioga kaasa android.util.Log teegiga ning on lihtsasti kasutatav. Esimese parameetrina antakse tavaliselt kaasa klassinimi, mida hoitakse klassiüleses muutujas nimega TAG, teise parameetrisse saab vastavalt vajadusele lisada muud informatsiooni (näide joonisel 22). Logimist kasutas autor rakenduse arendamise käigus selleks, et kontrollida, kas informatsioon jõudis soovitud kujul soovitud kohta, või harvem, et näha, millised meetodid käivitusid.

```
Log.d(TAG, "setUpRecyclerView: " + profiles)
```

Joonis 22. Näide logimisest Kotlinis.

Muudatusi testis autor jooksvalt Android Studio emulaatoril, proovides läbi veaohtlikke olukordi, ning vajadusel viis sisse parandusi. Samuti andis emulaatorite kasutamine parema ettekujutuse sellest, kuidas rakendus seadme peal välja võiks näha, mille põhjal sai muuta sobivamaks tekstisuuruseid. Android Studio emulaatori kasutamine on kiirem ja mugavam kui päris seadmega ühendamine ning võimaldab rakendust testida seadmetel, millele arendajal ligipääsu pole. Toiteploki ja seadme vahelist ühendust plaanib autor testida ka füüsilisi Android-seadmeid kasutades. Joonisel 23 on kõrvuti näidatud profiili redigeerimise vaade kolmel erineval emuleeritud seadmel: nutitelefonidel Nexus S ja Pixel 3a ning tahvelarvutil Nexus 7.

Joonis 23. Erinevad seadmed Android Studio emulaatoris.

Reaalseid andmeid autor testida ei saanud, kuna polnud toiteplokki, mis neid talle saadaks ega andmete vastuvõtmise loogikat. Seevastu pärines kogu rakenduses kuvatav test info JSON-failidest, mille sisu võiks jäljendada informatsiooni, mida toiteplokk tulevikus saatma hakkab. Seniks on võimalik muuta JSON-faile ja/või luua rakendusele näiteks seadmete vahelise suhtluse loogika kontrollimiseks automaatsed testid, kasutades selleks näiteks JUnit 4 raamistikku.

5 Kokkuvõte

Antud töö käigus osales autor programmeeritava toiteploki kasutajaliidese planeerimises, lõi disaini ja alustas rakenduse arendust. Planeerimis- ja disainiprotsessi kooskõlastati jooksvalt toiteploki inseneriga. Kasutajaliides jookseb Android operatsioonisüsteemiga mobiilsetel seadmetel. Autori valitud keeleks oli Kotlin, milles arendamist toetas Android Studio keskkond. Töö käigus valmistas autor esialgse graafilise prototüübi UXPin veebikeskkonnas ning alustas selle põhjal rakenduse arendamist. Rakendus võimaldab esialgu teisendada JSON-formaadis algandmeid vaadetele sobivale kujule, neid kuvada, nende põhjal navigatsiooni erinevatele vaadetele ja vastavalt andmetüüpidele kuvada erinevaid komponente.

Kuigi algselt oli eesmärk võimaldada toiteplokiga suhtlust üle Bluetooth Low Energy tehnoloogia, siis ajapiirangu ja seadme enda valmisise hilinemise tõttu, seda antud töö raames ei tehtud. Sellegipoolest plaanib autor rakendust edasi arendada ning on arvestanud, et hiljem võivad andmed tulla erinevaid kanaleid pidi, kas üle Bluetooth Low Energy tehnoloogia või näiteks üle interneti serverist.

Lisaks tavapärasele voolu ja pinge kontrollimisele, teeb antud lahenduse unikaalseks võimalus juhtmevabalt toiteploki tööd graafiliselt programmeerida. Selleks on tehtud algne konfigureerimislahendus, kuhu on võimalik peateljest sõltuvalt lisada koordinaate eri karakteristikutele (nagu näiteks voolutugevus ja pinge piirang). Kuigi hetkel pole võimalik peatelje liiki muuta ja näidisandmete põhjal on vaikimisi väärtus aeg, suudab rakenduse loogika hakkama saada ka teiste karakteristikutega, võimaldades tulevikus kasutajale suuremat paindlikkust. Android-seadmete laialdase leviku tõttu võiks see lahendus olla laialdaselt kättesaadav, piiratud eelkõige sellest, kas kasutajal on ligipääs spetsiifilisele toiteplokile.

Esialgu on edasi arendamise puhul põhifookuseks võimaldada suhtlus antud toiteplokiga ning tema graafilise programmeerimise võimaldamine lisaks punktide koordinaatidele visuaalse graafiku abil. Samuti on rakenduses juba jäetud koht seadete ja statistika

vaatamise jaoks ning korraga mitme kanali töö alustamise jaoks. Lisaks on soov võimaldada tingimuste lisamist näiteks töö käigus profiilitüübi vahetamise jaoks ning uute profiilitüüpide lisamine, näiteks akude laadimise jaoks. Rakendus vajab praegusest laialdasemat testimist erinevate ekraani suurustega ja toiteplokiga suhtlemisel, mida pole veel saanud teha.

Kasutatud kirjandus

[1] Cprime, „What is AGILE - What is SCRUM - Agile FAQs,“ [Võrgumaterjal]. Available: https://www.cprime.com/resources/what-is-agile-what-is-scrum/. [Kasutatud 10. märts 2020].

[2] M. Pinola, „Bluetooth Basics,“ 2. veebruar 2020. [Võrgumaterjal]. Available: https://www.lifewire.com/what-is-bluetooth-2377412. [Kasutatud 10. märts 2020].

[3] W3Schools, „JSON - Introduction,“ [Võrgumaterjal]. Available: https://www.w3schools.com/js/js_json_intro.asp. [Kasutatud 2. mai 2020].

[4] Lexico.com, „user interface,“ [Võrgumaterjal]. Available: https://www.lexico.com/definition/user_interface. [Kasutatud 2. mai 2020].

[5] V. Beal, „application (application software),“ [Võrgumaterjal]. Available: https://www.webopedia.com/TERM/A/application.html. [Kasutatud 2. mai 2020].

[6] PCMag, „timeout,“ [Võrgumaterjal]. Available: https://www.pcmag.com/encyclopedia/term/timeout. [Kasutatud 10. märts 2020].

[7] Dave, „What is a Power Supply and Types of Power Supply for Electrical Circuits,“ 10 12 2018. [Võrgumaterjal]. Available: https://www.watelectrical.com/what-is-a-power-supply-and-types-of-power-supply-for-electrical-circuits/. [Kasutatud 7. märts 2020].

[8] J. Martindale, „What is Wi-Fi? Here’s everything you need to know,“ 20. veebruar 2020. [Võrgumaterjal]. Available: https://www.digitaltrends.com/computing/what-is-wi-fi/. [Kasutatud 10. märts 2020].

[9] D. Pogue, „What Wi-Fi Stands for—and Other Wireless Questions Answered,“ 1. mai 2012. [Võrgumaterjal]. Available: https://www.scientificamerican.com/article/pogue-what-wifi-stands-for-other-wireless-questions-answered/. [Kasutatud 6. mai 2020].

[10] M. BRAIN, T. V. WILSON ja B. JOHNSON, „How WiFi Works,“ HowStuffWorks.com, 30. aprill 2001. [Võrgumaterjal]. Available: https://computer.howstuffworks.com/wireless-network2.htm. [Kasutatud 2. mai 2020].

[11] STMicroelectronics, „STM32WB55CG,“ [Võrgumaterjal]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32wb55cg.html. [Kasutatud 7. märts 2020].

[12] T&M Atlantic, „AKTAKOM Power Manager (APM),“ [Võrgumaterjal]. Available: https://www.tmatlantic.com/software/index.php?SECTION_ID=333&ELEMENT_ID=15816. [Kasutatud 7. märts 2020].

[13] Atkakom, „AKTAKOM Smart Data Logger,“ [Võrgumaterjal]. Available: https://play.google.com/store/apps/details?id=com.aktakom.asdl&hl=en. [Kasutatud 7. märts 2020].

[14] C. Hsiung, „VoltBot - Graphical Charger & DC Power Supply,“ [Võrgumaterjal]. Available: https://www.indiegogo.com/projects/voltbot-graphical-charger-dc-power-supply#/. [Kasutatud 7. märts 2020].

[15] Keysight Technologies, „E36300 Series Triple Output Power Supply,“ [Võrgumaterjal]. Available: https://www.keysight.com/zz/en/products/dc-power-supplies/bench-power-supplies/e36300-series-triple-output-power-supply-80-160w.html. [Kasutatud 7. märts 2020].

[16] J. Lee, „MakeUseOf,“ 6. märts 2017. [Võrgumaterjal]. Available: https://www.makeuseof.com/tag/build-android-app-programming-languages/. [Kasutatud 7. märts 2020].

[17] A. Sinicki, „I want to develop Android apps — What languages should I learn?,“ Android Authority, 10. august 2019. [Võrgumaterjal]. Available: https://www.androidauthority.com/develop-android-apps-languages-learn-391008/. [Kasutatud 7. märts 2020].

[18] AltexSoft, „The Good and The Bad of Xamarin Mobile Development,“ 26. aprill 2019. [Võrgumaterjal]. Available: https://www.altexsoft.com/blog/mobile/pros-and-cons-of-xamarin-vs-native/. [Kasutatud 7. märts 2020].

[19] U. Sehar, „Kotlin vs. React Native: Which Is Better?,“ DZone, 18. märts 2018. [Võrgumaterjal]. Available: https://dzone.com/articles/kotlin-vs-react-native-which-is-better. [Kasutatud 7. märts 2020].

[20] SPEC INDIA, „Java vs Kotlin: It’s Time To Expand Android Development,“ 8. jaanuar 2019. [Võrgumaterjal]. Available: https://www.spec-india.com/blog/java-vs-kotlin-its-time-to-expand-android-development. [Kasutatud 7. märts 2020].

[21] M. Woolley, „Developer Study Guide: An introduction to Bluetooth Low Energy Development,“ Bluetooth SIG, 2019.

[22] C. Y. Ong, „Punch Through,“ 10. september 2019. [Võrgumaterjal]. Available: https://punchthrough.com/android-ble-development-tips/. [Kasutatud 7. märts 2020].

[23] „Power Supply User Interface – UXPin preview,“ [Võrgumaterjal]. Available: https://preview.uxpin.com/4c9e837b0ef1bba68c0adfca1ccbbbdf9e6c7e97#/pages/126619289/simulate/no-panels?mode=i. [Kasutatud 6. mai 2020].

[24] A. Kadakmaa, „PSUInterface,“ [Võrgumaterjal]. Available: https://github.com/aikada/PSUInterface. [Kasutatud mai 6. 2020].

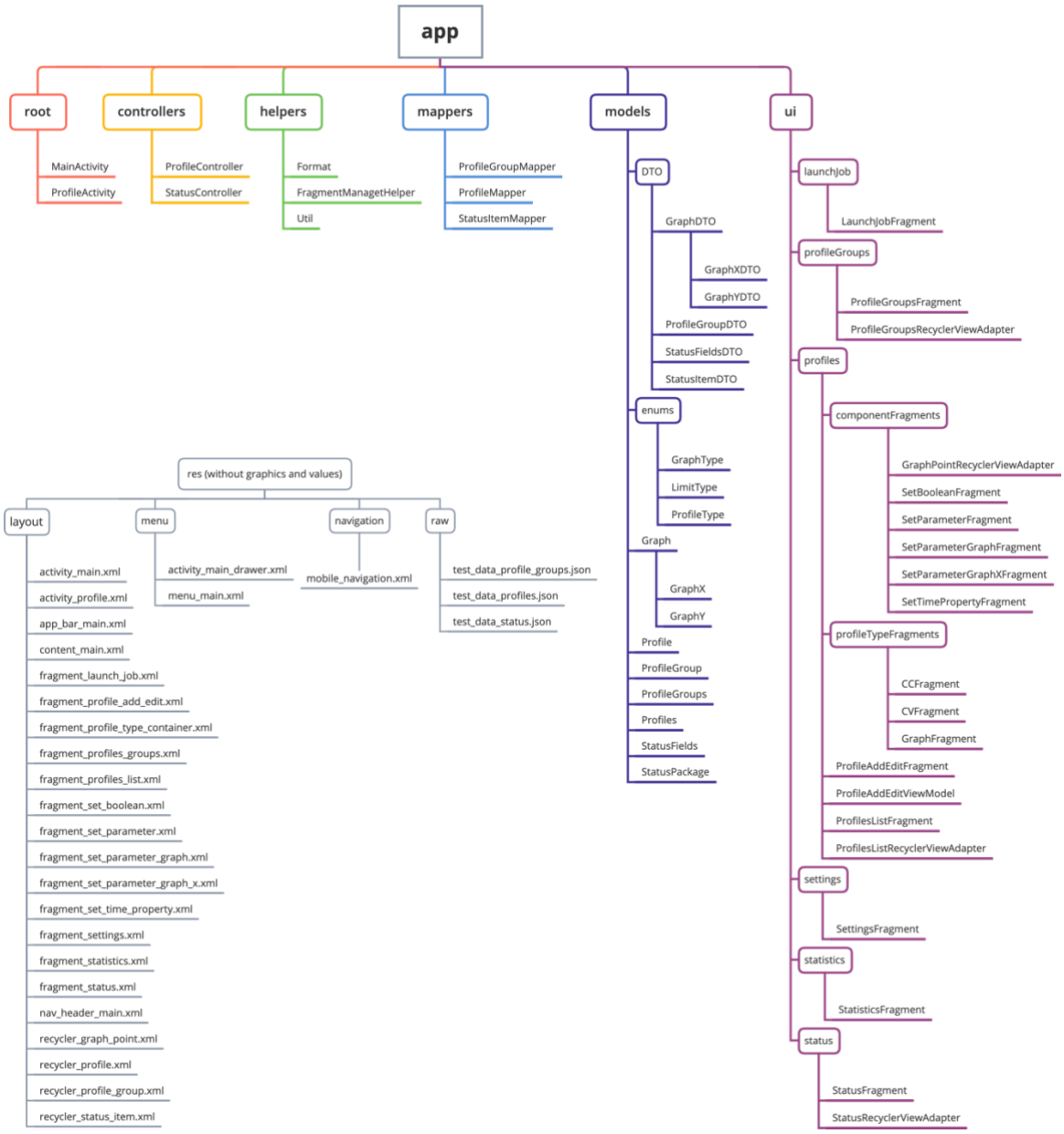
[25] „Kotlin multiplatform / multi-format reflectionless serialization,“ [Võrgumaterjal]. Available: https://github.com/Kotlin/kotlinx.serialization. [Kasutatud 27. aprill 2020].

Lisa 1 – Veebilingid rakenduse graafilisele prototüübile ja programmikoodile

Antud lõputöö raames loodud graafiline prototüüp UXPin veebikeskkonnas on leitav järgnevalt veebiaadressilt:
https://preview.uxpin.com/4c9e837b0ef1bba68c0adfca1ccbbbdf9e6c7e97#/pages/126619289/simulate/no-panels

Programmikood on leitav GitHubist järgnevalt veebiaadressilt: https://github.com/aikada/PSUInterface

Lisa 2 – Rakenduse arhitektuur



Lisa 3 – Profiilide näidisandmed JSON-kujul


```
{
  "profiles": [
    {
      "name": "profile 1",
      "group": "profile group 1",
      "type": "Graph",
      "duration_ms": null,
      "graph": {
        "X": {
          "value": "time",
          "offset": 0,
          "scale": 5
        },
        "Y": [
          {
            "value": "voltage",
            "offset": 0,
            "scale": 1,
            "points": [
              [
                0,
                0
              ],
              [
                1000,
                10
              ]
            ]
          },
          {
            "value": "current limit",
            "offset": 0,
            "scale": 1,
            "points": [
              [
                0,
                10
              ],
              [
                1000,
                10
              ],
              [
                1001,
                1
              ]
            ]
```

```
      },
      {
        "value": "resistance",
        "offset": 0,
        "scale": 1,
        "points": [
          [
            0,
            1
          ]
        ]
      }
    ]
  },
  "jumps": [
    {
      "type": "if",
      "target": {
        "group": "profile group 1",
        "name": "profile 2"
      },
      "value": "power",
      "comp": ">",
      "limit": 10.0
    },
    {
      "type": "while",
      "target": {
        "group": "profile group 1",
        "name": "profile 3"
      },
      "value": "ADC1",
      "comp": ">",
      "limit": 0.333
    }
  ]
},
{
  "name": "profile 2",
  "group": "profile group 1",
  "type": "CV",
  "duration_ms": null,
  "value": 10,
  "limitValue": 2,
  "resistance": 0,
  "jumps": [
  ]
},
{
  "name": "profile 3",
  "group": "profile group 1",
```

```
      "type": "CC",
      "duration_ms": 1234567,
      "value": 10,
      "limitValue": 2,
      "resistance": 2,
      "latchOff": true
    }
  ]
}
```