TALLINNA TEHNIKAÜLIKOOL
Tartu kolledž

ANDROID RAKENDUSE LOOMINE TARTU KOLLEDŽI INFOEKRAANI KUVAMISELE

ANDROID APPLICATION DEVELOPMENT FOR THE INFOSCREEN IN TARTU COLLEGE

RAKENDUSKÕRGHARIDUSTÖÖ

Üliõpilane: Indrek Arumets

Üliõpilaskood 193002EDTR

Juhendaja: Taavi Kase, insener

Tartu 2024

(Tiitellehe pöördel)

AUTORIDEKLARATSIOON

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

01.2024

Autor: Indrek Arumets / allkirjastatud digitaalselt /

Töö vastab bakalaureusetöö esitatud nõuetele

01.2024

Juhendaja: Taavi Kase /allkirjastatud digitaalselt /

Kaitsmisele lubatud

01.2024

Kaitsmiskomisjoni esimees: Aime Ruus / allkirjastatud digitaalselt /

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Indrek Arumets (sünnikuupäev: 01.04.2000)

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose

„ANDROID RAKENDUSE LOOMINE TARTU KOLLEDŽI INFOEKRAANI KUVAMISELE“,

mille juhendaja on Taavi Kase,

1.1 reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

1.2 üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.

3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

---

[1] *Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil.*

/ allkirjastatud digitaalselt /

01.2024

TalTech Instituudi nimetus

LÕPUTÖÖ ÜLESANNE

Üliõpilane: Indrek Arumets, 193002EDTR
Õppekava, peaeriala: EDTR17/21, Arukad süsteemid ja rakendusinfotehnoloogia
Juhendaja(d): Insener, Taavi Kase, 6204808

Lõputöö teema:
(eesti keeles) Android rakenduse loomine Tartu kolledži infoekraani kuvamisele
(inglise keeles) Android application development for the infoscreen in Tartu College

Lõputöö põhieesmärgid:

1. Arendada Android operatsioonisüsteemil baseeruv mobiilirakendus sisekliima ning tunniplaani efektiivsemaks jälgimiseks.
2. Luua võimalus kasutada rakendust õppevahendina.

Lõputöö etapid ja ajakava:

Nr	Ülesande kirjeldus	Tähtaeg
1.		
2.		
3.		

Töö keel: Eesti keel **Lõputöö esitamise tähtaeg:** 8. jaanuar 2024 a

Üliõpilane: Indrek Arumets 8. jaanuar 2024 a
/ allkirjastatud digitaalselt /

Juhendaja: Taavi Kase 8. jaanuar 2024 a
/ allkirjastatud digitaalselt /

Programmijuht: Aime Ruus 8. jaanuar 2024 a
/ allkirjastatud digitaalselt /

Kinnise kaitsmise ja/või lõputöö avalikustamise piirangu tingimused formuleeritakse pöördel

SISUKORD

EESSÕNA

Käesoleva lõputöö teema sõnastas Tallinna Tehnikaülikooli Tartu kolledži insener Taavi Kase. Samuti toetas tema Android mobiilirakenduse valmimist ning andis põhjalikku tagasisidet kogu arenduse protsessi vältel.

Eluslabor, sisekliima, õppematerjal, mobiilirakendus, bakalaureusetöö.

LÜHENDITE JA MÕISTETE LOETELU

Android – Operatsioonisüsteem mobiilsetele seadmetele.

Android Studio – Arenduskeskkond, mis toetab Android operatsioonisüsteemil baseeruvate mobiilirakenduste arendamist.

API – *Application Programming Interface* on rakendustarkvara liides, mis suunab ühest tarkvararakendusest tulnud päringud või andmed edasi teise programmi. Võimaldab seejuures seadmete või rakenduste vahel vahetada informatsiooni või käsklusi.

Bluetooth – Sideprotokoll, mis edastab raadiolainetena krüpteeritud kujul andmeid erinevate seadmete vahel.

Figma – Pilvepõhine disainitööriist mobiili- ja veebirakenduste kujundamiseks.

JSON – Andmevahetus formaat ehk *JavaScript Object Notation*, mis baseerub *JavaScript* süntaksil.

Kotlin – Programmeerimiskeel, mis on projekteeritud ühilduma Java programmeerimiskeelega.

WiFi – *Wireless Fidelity* on traadita sideprotokoll, mille töösagedus on vahemikus 2-5GHz.

XML – Märgendkeel, mida kasutatakse struktureeritud andmete esitamiseks arvutisüsteemides.

SISSEJUHATUS

Alates 2015. aastast tegeleb Tartu kolledži küberfüüsikaliste süsteemide töörühm Eluslaboratooriumi arendamisega A-korpuses. Antud keskkond on tudengitele küberfüüsikaliste süsteemide uurimiseks sobiv aluspinnas. Tallinna Tehnikaülikooli Tartu kolledži Eluslaboratooriumis on fookus suunatud hooneautomaatika ning küttesüsteemi juhtimise edasiarendustele. Automaatikasüsteem võimaldab talletada MeiePilv-nimelisse pilveplatvormi, edaspidi nimetatud „MeiePilv" andmebaasi sisekliima andmeid, seadeparameetreid, süsteemi seisundeid ning Tartu kolledži ilmajaama informatsiooni. [1]

Tartu kolledži andmebaasi salvestatud info põhjal on võimalik analüüsida energiakasutust. Tuginedes andmetele saab projekteerida lahendusi, mis optimeerivad energiakulu. Eluslaboratooriumi arendamise käigus on õppehoonesse paigaldatud infoekraan, mis kuvab A-korpuse ruumide nimekirja ning sisekliima infot. Andmeid – õhu suhteline niiskus, süsihappegaasi sisaldus, ruumi- ning seadetemperatuur - on võimalik vaadelda õpperuumide seintel asuvatelt ruumikontrollerite LCD-ekraanidelt. [1]

Selline informatsiooni visualiseerimise lahendus eeldab igas A-korpuse õpperuumis asuva ruumikontrolleri jälgimist individuaalselt. MeiePilves on võimalik vaadelda ruumide sisekliima andmeid terviklikult ühes andmebaasis. Seejuures ei ole kirjeldatud võimekus mugav ning kiire, sest see nõuab kasutajalt MeiePilve ligipääsemiseks VPN-i olemasolu.

Käesoleva lõputöö ülesanne on luua Tallinna Tehnikaülikooli Tartu kolledžisse Android operatsioonisüsteemil põhinev mobiilirakendus, mis kuvab A-korpuse õpperuumide sisekliima andmeid. Seejuures võimaldab vaadata tunniplaani vastavalt kursusele ja ainekavale. Samuti keskendutakse töö käigus kahele alaülesandele: luua võimekus tulevikus jälgida kampuse parklas olevate sõidukite hulka ning kasutusele võtta kasutaja registreerimise ja sisselogimise süsteem.

Lõputöö üks eesmärk on rakendada Android operatsioonisüsteemil baseeruvat mobiilirakendust sisekliima ning tunniplaani efektiivsemaks jälgimiseks. Rakendus kuvab kasutajale tunniplaaniga seonduvat ning järgmisi parameetreid: temperatuur, süsihappegaasi ning niiskusetase. Teine eesmärk on luua võimalus kasutada rakendust õppevahendina. See annab tudengitele uudse õppesuuna Tartu kolledži Eluslaboratooriumi projektiainetes ning lõputööde teostamiseks, mille käigus mobiilirakendust edasi arendada.

Android operatsioonisüsteemil baseeruv mobiilirakendus arendatakse Android Studio keskkonnas. Programmeerimiseks rakendatakse programmeerimiskeelt Kotlin. Rakenduse esmane väljanägemine kujundatakse pilvepõhises disainiplatvormis Figma ning testserverina kasutatakse WAMP-serverit. Edasised disaini muudatused teostatakse Android Studio keskkonnas XML-märgendkeeles.

Käesoleva lõputöö teoreetilises osas kirjeldab autor lähteülesande püstitust ning sellest tulenevaid ülesandeid. Seejärel käsitleb Tartu kolledži Eluslaboratooriumi tähtsust, arhitektuuri ning seadmeid sisekliima seireks. Samuti annab autor ülevaate MeiePilve serveri ülesehitusest ja olulisusest ning toob esile kaks mobiilirakendust sisekliima jälgimiseks. Töö praktilises osas selgitab autor arenduskeskkonna ülesseadmist ning süsteemi miinimumnõudeid Android Studio kasutamiseks. Lisaks käsitleb töö käigus rakendatud tööriistu ning testservereid. Ühtlasi kirjeldab rakenduse arendusprotsessi ning esitab tulemuste analüüsi.

Lõputöö on kirjutatud eesti keeles ja sisaldab teksti 43 leheküljel, 10 peatükki ning 28 joonist.

1. LÄHTEÜLESANNE

Käesolevas peatükis annab autor ülevaate lõputöö lähteülesande püstitusest, loodavatest funktsioonidest ning arendusprotsessi etappidest. Samuti tutvustab, milline on loodava tarkvara väärtus Tallinna Tehnikaülikooli Tartu kolledži Eluslaboratooriumile.

1.1 Ülesande püstitus

Lõputöö käigus arendatakse Android operatsioonisüsteemil baseeruv mobiilirakendus Tartu kolledži A-korpuse õpperuumide sisekliima jälgimiseks. Loodav tarkvara kasutab MeiePilve andmebaasist andmete allalaadimiseks kampuse rakendustarkvara liidest InfoAPI. Selline lahendus võimaldab Eluslaboratooriumi automaatikasüsteemi seadmetel ning programmidel saata andmeid või päringuid API-sse, mis suunab need edasi MeiePilve. Arendatav rakendus omab võrguühilduvust ja edastab kasutajale järgmisi sisekliima andmeid: temperatuur, süsihappegaasi hulk ning niiskustase. Seejuures loob autor funktsionaalsuse jälgida tunniplaani vastavalt ainekavale ja kursusele. Rakenduse disainimisel on võetud arvesse võimalikke edasiarendusi- luua sisselogimise süsteem ja kuvada parklas olevate sõidukite arvu.

Valmivat mobiilirakendust saab tulevikus kasutada Eluslaboratooriumi õppevahendina. Üks suund on sisekliima andmete jälgimine ja kogumine loodava tarkvara abil. Samuti on võimalik tutvuda tarkvaraarendusega ning teostada edasiarendusi projektiaine või lõputööde raames.

1.2 Lähteülesandest tulenevad ülesanded

Lõputöö käigus teostatavad ülesanded ning nõutavad tulemused on lähtuvalt lähteülesandest järgmised:

- Töötada läbi materjalid ning Android Studio arenduskeskkonnale orienteeritud dokumentatsioon.
- Paigaldada arenduseks kasutatavasse seadmesse Android Studio arenduskeskkond.
- Seadistada Wamp testserver mobiilirakenduse andmevoo testimiseks.
- Seada üles töökeskkond ning lõputöö arendusprojekt.
- Disainida Android mobiilirakenduse leheküljed ning kasutajavoog.

- Seada üles repositoorium versioonihalduskeskkonnas GitLab.
- Programmeerida Android mobiilirakendusse võrguga ühildumise võimekus.
- Programmeerida Tartu kolledži InfoAPI-ga suhtlemise ja päringute tegemise funktsionaalsus.
- Vormistada rakenduses ruumide nimekirja kuvamine.
- Luua vastavalt ruuminumbrile viimaste sisekliima andmete (temperatuur, süsihappegaas, õhuniiskus) kuvamise funktsionaalsus.
- Programmeerida tunniplaani kuvamine rakenduses vastavalt ainekavale ning kursusele.
- Disainida rakendusse Tartu kolledži parklas olevate sõidukite arvu kuvamine.
- Disainida rakendusse kasutajate loomise ja sisselogimise süsteem.

2. ELUSLABOR

Käesolevas peatükis annab lõputöö autor ülevaate Tartu kolledži Eluslabori tähtsusest, arhitektuurist ning komponentidest. Seejuures käsitleb autor sisekliima automaatika juhtimise tasandeid. Samuti, kuidas toimub sisekliima andmete edastamine anduritest MeiePilve andmebaasi.

2.1 Eluslabori tähtsus

Eluslaboratoorium erineb tavapärasest laboratooriumist, kuna see ei keskendu üksikute protsesside uurimisele tehiskeskkonnas. Seejuures on Eluslaboratooriumis rohkem andureid ja kogutakse suuremas mahus andmeid. Käsitletavas keskkonnas toimub uute lahenduste testimine ning aset leidnud muutuste analüüsimine. Eluslaboratooriumi eelis seisneb KFS õppekavas turvalises keskkonnas eksperimentide ja uuringute teostamisel. Selline võimekus Tartu kolledžis toetab uuenduslike ideede ohutut arendamist ning katsetamist reaalsetes oludes. [2]

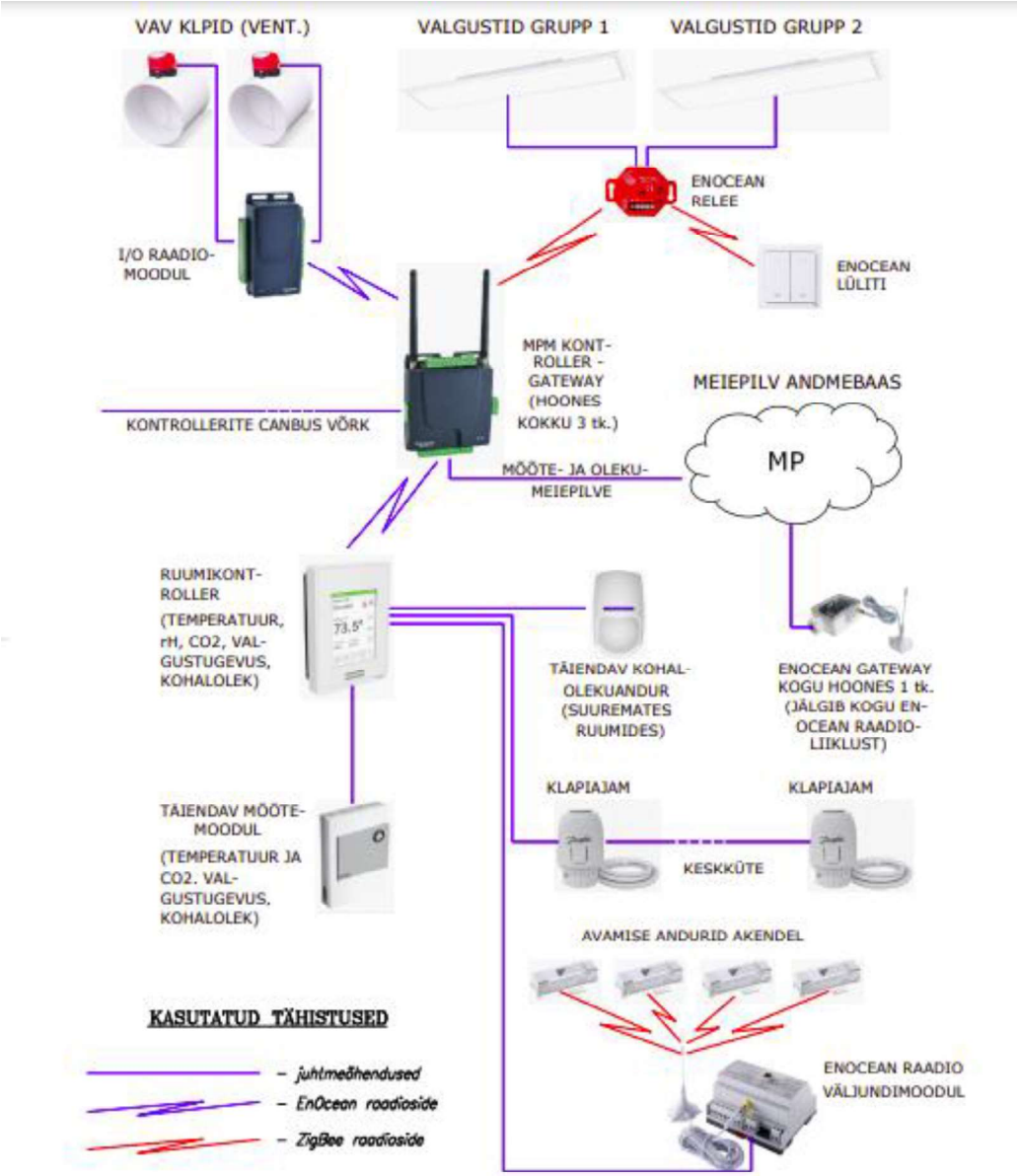
Tartu kolledži Eluslaboratoorium on eelkõige õpikeskkond, mis on suunatud KFS eriala tudengitele. Antud keskkonnas saab teostada praktikume ja katseid, mis aitavad mõista paremini küberfüüsikalise süsteemi olemust ning käitumist. Õppetöös on Eluslaboratooriumi fookus suunatud peamiselt Tartu kolledži sisekliimaautomaatikale ning selle juhtimisele. Selliselt on tagatud võimalikult efektiivne energiakasutus. [2] Hooneautomaatika protsesside jälgimisel ning sisekliima andmete analüüsimisel saavad tudengid praktilisi kogemusi. Saadud teadmisi on võimalik rakendada automaatikasüsteemide ning sisekliimat juhtivate tehniliste lahenduste arendamisel. [3]

2.2 Eluslabori arhitektuur

Tartu kolledži Eluslaboratoorium koosneb järgmistest komponentidest (Joonis 2.1):

- kontaktandurid akendel,
- Enocean raadioväljundimoodul,
- keskkütteradioaatorite klapiajamid,
- täiendav mõõtemoodul (temperatuur ning süsihappegaas),
- Schneider Electric SE8350 ruumikontroller (temperatuuri, süsihappegaasi ja valgustugevuse tase ning ruumis kohalolek),

- täiendav kohalolekuandur (suuremates ruumides),
- Schneider Electric MPM lüüsikontroller,
- Enocean relee ning lüliti (valgustuse lülitamiseks),
- Enocean moodul (Enocean raadioliikluse jälgimiseks kogu hoones).



Joonis 2.1 Eluslaboratooriumi seadmestik

Eluslaboratooriumi juhtimistasand jaguneb kaheks. Madalama taseme sisekliima automaatika juhtimiseks kasutatakse Schneider Electric SE8350 ruumikontrollereid. Kõrgema taseme jaoks rakendatakse sama ettevõtte toodetud MPM lüüsikontrollereid.

[2] Samaaegselt tegelevad madalama astme ruumikontrollerid sisekliima andmete kogumisega, mida edastavad ZigBee raadioprotokolli kaudu MPM kontrollerile. Seejärel edastab MPM kontroller MeiePilve andmebaasi järgmised sisekliima ning seadmeoleku parameetrid:

- temperatuuri tase,
- seadetemperatuuri tase,
- õhuniiskuse tase,
- süsihappegaasi tase,
- valgustugevuse tase,
- kontaktandurite olek akendel,
- hõivatuse olek ruumis,
- küttesüsteemi olek ruumis.

2.3 Eluslaboratooriumi seadmed sisekliima seireks

Tartu kolledži Eluslaboratooriumi sisekliima monitoorimiseks kasutataval Schneider SE8350 (Joonis 2.2) ruumikontrolleril on mitmeid tehnoloogilisi eeliseid. Antud kontrollerit on võimalik programmeerida vastavalt kasutaja vajadustele. Samuti on tootja lisanud seadmele LCD ekraani, mis kuvab reaalajas sisetemperatuuri, õhuniiskust ning süsihappegaasi taset. Seejuures võimaldab SE8350 ruumikontroller edastada kasutajale infot ruumi sisekliimas toimunud muutuste kohta. Kui mõni näitaja on tõusnud üle normi piiri, vahetab LCD ekraan värvi. [2]

Lisaks Schneider SE8350 ruumikontrollerile on Eluslaboratooriumi sisekliima seires kasutusel sama tootja SRC210 (Joonis 2.3) täiendav mõõtemoodul. Antud mõõtemoodul toetab SE8350 kontrolleri tööd ning edastab sellele lisaandmeid temperatuuri, õhuniiskuse ning süsihappegaasi tasemest. [2]

Joonis 2.2 Schneider Electric SE8350 [4]

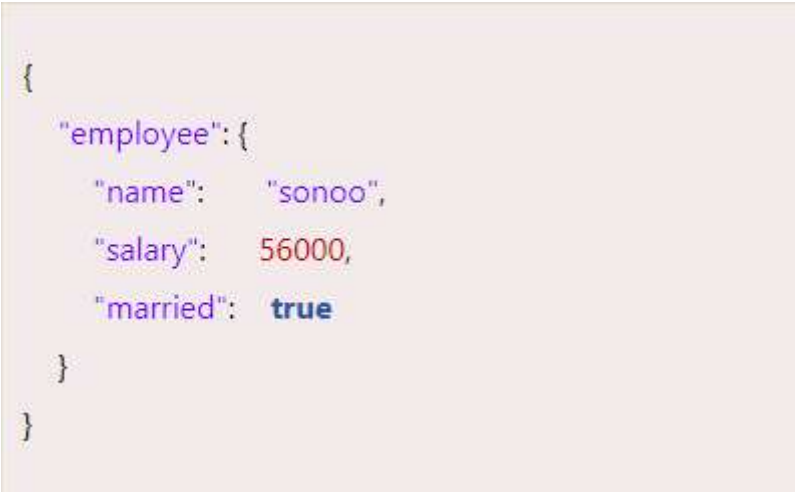
Joonis 2.3 Schneider Electric SCR210 [5]

3. PILVELABORI PLATVORM

Tartu kolledži pilvelabori platvorm võimaldab tudengitel rakendada mitmesuguseid seadmeid ning tehnoloogiad, mis on laialdaselt levinud asjade interneti ning tööstusautomaatika valdkonnas. Seejuures on antud platvormil võimekus juhtida õpperuumides asuvaid seadmeid. [6] Kaugjuhtimise teel on võimalik reguleerida küttesüsteemi selliselt, et see tagaks sobiva sisetemperatuuri õppetöö teostamiseks. Samuti võimaldab pilvelabori platvorm koguda andurite kaudu andmeid andmebaasi, mida on tulevikus võimalik analüüsida ning süsteemi arenduseks rakendada [6].

3.1 MeiePilve server

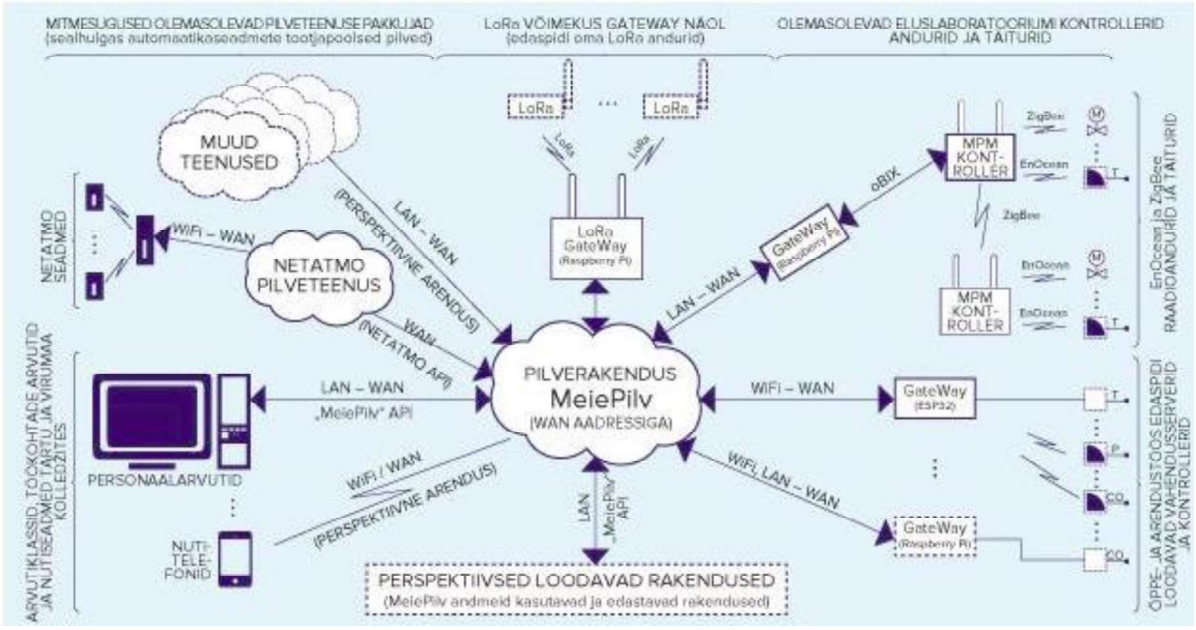
Tartu kolledži pilverakendus MeiePilv on MongoDB andmebaasil baseeruv server, mis omab API-t ning veebiliidest [6]. MongoDB on avatud lähtekoodiga dokumendipõhine NoSQL-andmebaas, mis kuulub mitterelatsiooniliste andmebaaside hulka [7]. Relatsioonilisest ehk SQL-andmebaasist erineb NoSQL selle poolest, et andmed ei ole omavahel relatsiooniliselt ühendatud tabelitesse. Andmete hoiustamine toimub JSON formaadis (Joonis 3.1) dokumentides ning päringute teostamiseks ei kasutata SQL-päringu keelt. Samuti ei vaja see kindlaks määratud andmemudelit, mis võimaldab andmeid talletada struktureerimata kujul. Sellise andmebaasi rakendamise vajadus tuleneb sellest, et Eluslaboratooriumi edasiarenduse käigus on keeruline ette näha, millisel kujul logisid hakatakse tulevikus esitama. [8]



Joonis 3.1 JSON formaadis dokument [9]

MeiePilve veebiliides võimaldab andmebaasi edastatavaid andmeid kuvada graafiku või tabeli kujul. Samuti saab teha andmepäringuid vastavalt tunnusele ning administreerida salvestusprotsessi. Serveri API toetab uute tehnoloogiliste lahenduste arendamist, mis on suunatud MeiePilve ja rakenduse vaheliseks andmeedastuseks. [6]

Tartu kolledži pilverakendusel MeiePilv on võimekus vastu võtta informatsiooni hooneautomaatika lüüsidelt, mikrokontrolleritega sõlmedelt ja teisaldatavatelt NetAtmo mõõtemoodulitelt ning LoRaNet moodulitelt. Samuti pakub MeiePilv Tartu kolledži sisevõrgu kaudu ligipääsu andmebaasile andmete analüüsimiseks, lisamiseks ning seadistamiseks. InfoAPI võimaldab uute seadmete ja mobiilirakenduste lisamist pilverakenduse süsteemi (Joonis 3.2).



Joonis 3.2 Tartu kolledži pilvelabori platvormi struktuur [6]

4. SARNASED MOBIILIRAKENDUSED

Käesolevas peatükis annab autor ülevaate kahest mobiilirakendusest, millega on võimalik lahendada lõputöös püstitatud eesmärke. Esile tuuakse rakendused uHoo ning Awair. Seejuures käsitletakse mobiilirakendustega ühilduvate seadmete maksumust, tööpõhimõtet ning pakutavaid funktsioone. Autor valis eelpool nimetatud seadmed lähtudes nende funktsionaalsusest, mõõdetavatest parameetritest ning haakumisest lõputööga.

4.1 Mobiilirakendus uHoo

Mobiilirakendus uHoo on võimalik paigaldada nii Android kui ka iOS operatsioonisüsteemil baseeruvatele nutiseadmetele. Esimesel kasutuskorral peab kasutaja looma rakenduses konto. Seejärel küsib tarkvara nõusolekut *Bluetooth* ühenduse loomiseks mobiilirakenduse ja õhumonitori vahel. Käesoleva seadme maksumus on 299$ ning pakub laialdaselt funktsioone, mis aitavad kasutajal kujundada oma kodu sisekliimat. Rakendus edastab teateid, kui mõne sisekliima parameetri tase tõuseb üle piirmäära. Samuti jagab soovitusi, kuidas tõsta ruumi õhukvaliteeti. Lisaks annab rakendus ülevaate temperatuuri, suhtelise õhuniiskuse, süsihappegaasi, tolmu osakeste, vingugaasi, õhurõhu, õhus leiduvate kemikaalide ning osooni tasemest (Joonis 4.1). [10]

Joonis 4.1 Mobiilirakendus uHoo ning õhumonitor [10]

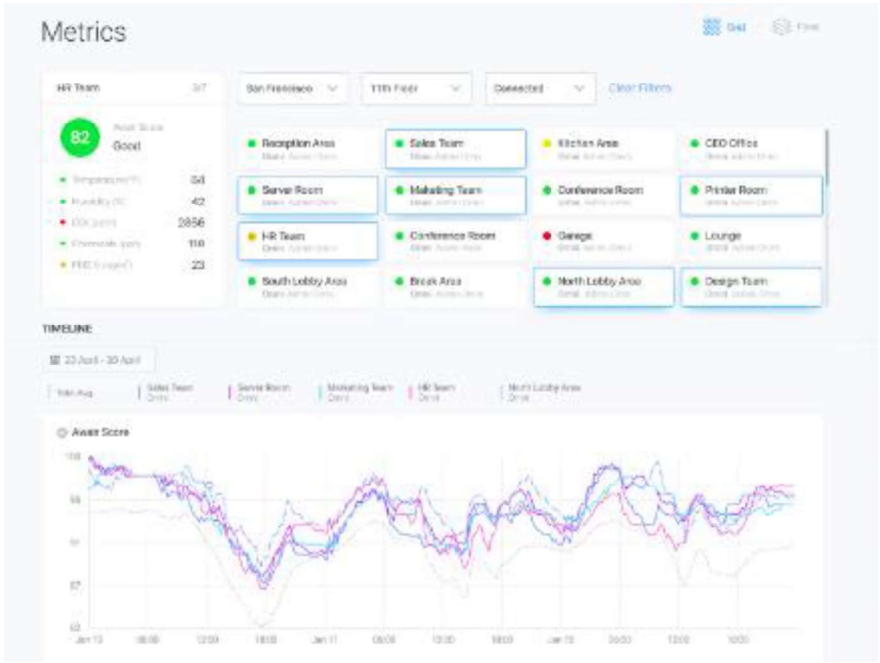
Mobiilirakenduse uHoo pakutavad funktsioonid võimaldavad lõputöös seatud eesmärki ellu viia - luua võimekus Tartu kolledži A-korpuse õpperuumide sisekliima jälgimiseks nutiseadmes. Seejuures on oluline juhtida tähelepanu sellele, et rakendus uHoo võimaldab jälgida põhjalikumalt ruumi õhukvaliteeti ning samuti neid parameetreid (vingugaasi, tolmu osakeste ning õhus leiduvate kemikaalide tase), mida praegused Eluslabori tehnilised lahendused ei paku. Siiski toob käesoleva lõputöö autor esile, et käsitletud lahendus on kulukas ja logistiliselt keeruline kasutada kõikides Tartu kolledži A-korpuse ruumides. Igale õpperuumile on vaja soetada eraldi õhumonitor, mis edastab kasutajale sisekliima andmeid. Seejuures nõuab iga seade eraldi kasutajakonto loomist mobiilirakenduses, mistõttu ei ole võimalik jälgida sisekliimat õpperuumides ühtse tervikuna. Samuti ei ole käsitletud lahendusega tulevikus võimalik juhtida Tartu kolledži kütte- ning ventilatsioonisüsteemi.

4.2 Mobiilirakendus Awair

Nutiseadmete operatsioonisüsteemidest toetavad mobiilirakenduse Awair installeerimist iOS ning Android. Esmasel kasutusel peab kasutaja looma konto ning suunduma seadme haldamise leheküljele. Seejärel luuakse ühendus mobiilirakenduse ning Awair seadme (Joonis 4.2) vahel. Õhumonitori maksumus on varieeruv sõltuvalt mitu seadet soovib ettevõte või kool enda hoonesse üles seada. Käsitletav õhumonitor on võimalik paigaldada mitmesse ruumi ning luua seeläbi seadmetest ühtne taristu. Antud süsteem võimaldab rakenduse juhtpaneelil (Joonis 4.3) kuvada reaalajas sisekliima andmeid kõikide ruumide kohta, kus paikneb Awair Omni seade. Mobiilirakendus suudab kasutajale edastada infot temperatuuri, õhuniiskuse, süsihappegaasi, õhus leiduvate kemikaalide, tolmu osakeste, heli ning valguse taseme kohta. [11]

Lähtudes mobiilirakenduse Awair funktsioonidest ning kasutusvõimalustest, sobib samuti antud tarkvara ning seade käesolevas lõputöös esile toodud eesmärgi teostamiseks. Awair Omni õhumonitoridest on võimalik luua Tartu kolledži A-korpuse õpperuumidesse süsteem, mis võimaldab jälgida sisekliimat tervikuna mobiilirakendusest. Siiski märgib autor, et käsitletud seadmel ning tarkvaral puudub võimekus juhtida kütte- ning ventilatsioonisüsteemi. Autori hinnangul on tulevikus sellise funktsionaalsuse arendamine käesoleva lõputöö käigus valmivas rakenduses teostatav.

Joonis 4.2 Awair Omni õhumonitor [11]



Joonis 4.3 Rakenduse Awair juhtpaneel [11]

5. ARENDUSKESKKONNA SEADISTAMINE

Käesolevas peatükis annab autor ülevaate Android operatsioonisüsteemist ning rakendusvõimalustest. Seejuures käsitletakse mobiilirakenduse arendamise juures vajalikku tarkvara ja selle tehnilisi miinimumnõudeid. Samuti kirjeldab autor Android Studio programmi installeerimist ning testseadme ja arendusprojekti seadistamist.

5.1 Android operatsioonisüsteem

Android on *Linux*'i tuumal baseeruv mobiilseadmetele suunatud operatsioonisüsteem. Platvorm on avaldatud Apache versiooni 2.0 avatud lähtekoodi litsentsi põhjal, mis võimaldab arendajatel luua erinevaid versioone Android operatsioonisüsteemist mitmesugustele koduseadmetele. [12]

Kasutajaliides põhineb operatiivsel käsitsemisel, mille juures käivitatakse funktsioone seadme ekraani puudutamisel või libistamisel. Interaktsioonide käigus annab Android operatsioonisüsteem kasutajale tagasisidet vibratsioonide kaudu. Nutiseadme käivitusel kuvatakse koduekraani, rakenduste ikoone ning teavet seadme võrguühendusest. Samuti on operatsioonisüsteem optimeeritud toiteaku, protsessori ning mälu säästlikule kasutamisele. Seejuures peatatakse mitteaktiivsed protsessid. [12]

Android operatsioonisüsteem toetab mitmeid andmeside- ja mobiilsidestandardeid. Standardide hulgas on enim kasutusel *Bluetooth*, GSM/HSDPA, CDMA/EV-DO, WiFi ning 3G sideprotokollid. Samuti rakendatakse Android operatsioonisüsteemi paljudes digitaalseadmetes – GPS, kiirendusmõõtjad, kompassid ning fotokaamerad. [12]

5.2 Android Studio nõuded seadmele

Android operatsioonisüsteemil baseeruva mobiilirakenduse arendamisel on oluline, et rakenduse ülesehitusel kasutatav seade on piisavalt võimekas. Android Studio platvorm on Android rakenduste loomisel levinuim arenduskeskkond. See pakub arendajale põhjalikku dokumentatsiooni ning toetavat kogukonda. Arenduskeskkonda on võimalik installeerida järgmistele operatsioonisüsteemidele: *Windows*, *MacOS* ning *Linux*. Järgnevalt kirjeldab autor seadmete tehnilisi miinimumnõudeid Android Studio platvormi paigaldamisel [13].

Windows

- Microsoft Windows 8 (64-bit)
- Operatiivmälu 8 GB
- Protsessor, mis kasutab x86_64 arhitektuuri või 2. põlvkonna Intel Core/AMD protsessor
- Põhimälu 8 GB
- Ekraani resolutsioon 1280 x 800

MacOS

- MacOS 10.14 (Mojave)
- Operatiivmälu 8 GB
- Protsessor, mis kasutab M1 kiipi või 2. põlvkonna Intel Core protsessor
- Põhimälu 8 GB
- Ekraani resolutsioon 1280 x 800

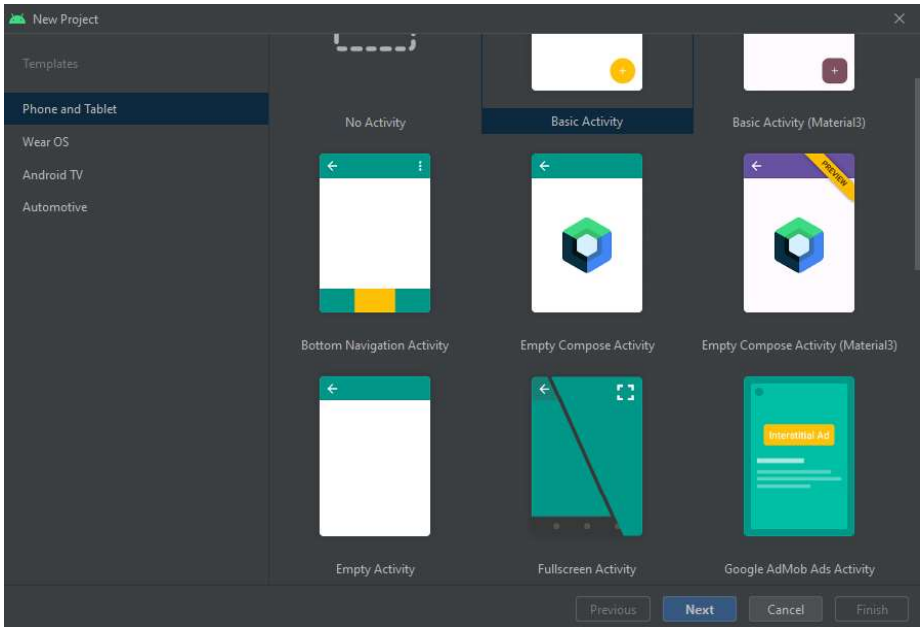
Linux

- Linux distributsioon (64-bit), mis toetab *Gnome*'i, KDE või Unity DE-d
- Operatiivmälu 8 GB
- Protsessor, mis kasutab x86_64 arhitektuuri või 2. põlvkonna Intel Core/AMD protsessor
- Põhimälu 8 GB
- Ekraani resolutsioon 1280 x 800

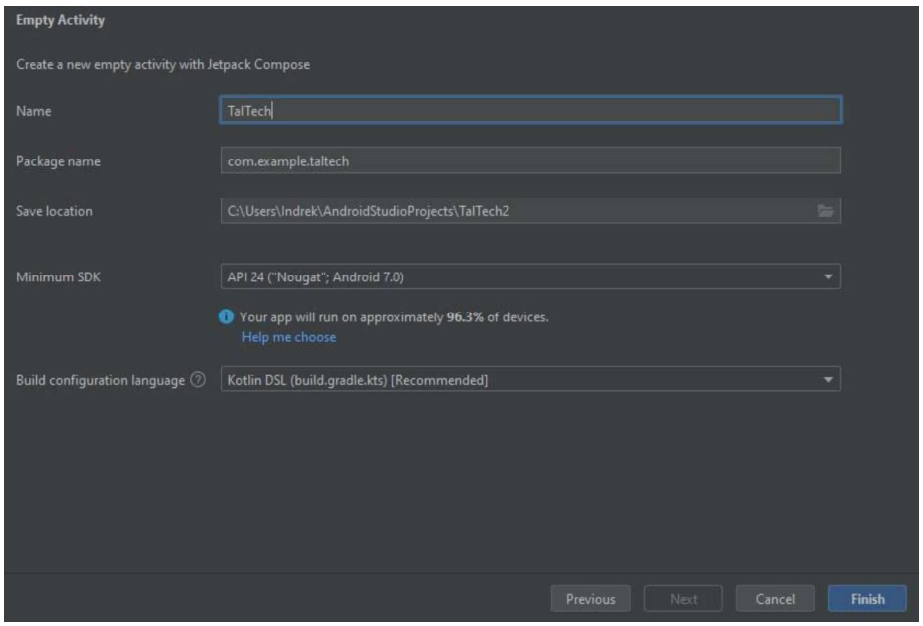
5.3 Projekti ülesseadmine

Android mobiilirakenduse arendamiseks kasutatav Android Studio arenduskeskkond on võimalik allalaadida ametlikult veebilehelt https://developer.android.com/studio/install. Allalaadimise järgselt tuleb alustada programmi installeerimisega ning nõustuda kasutustingimustega.

Järgmise sammuna peab kasutaja valima Android Studio platvormil arendusmalli, millisele seadmele soovitakse rakendust arendada. Valikus on mobiili-, tahvelarvuti-, teleri- ning autotööstusmall (Joonis 5.1). Järgnevalt valib rakenduse arendaja, millise tegevuse raamistikku soovib projekti ülesseadmisel rakendada. Seejärel annab kasutaja projektile nime ja valib asukoha, kuhu soovib oma faili salvestada (Joonis 5.2). Versioonis 2022.3.1 pakub Android Studio vaikimisi programmeerimiskeelena Kotlinit. Minimaalse SDK versioonina soovitatakse kasutajale API 24, mis võimaldab rakendusel töötada kuni 96.3% seadmetel.



Joonis 5.1 Android Studio projekti arendusmall ja raamistik

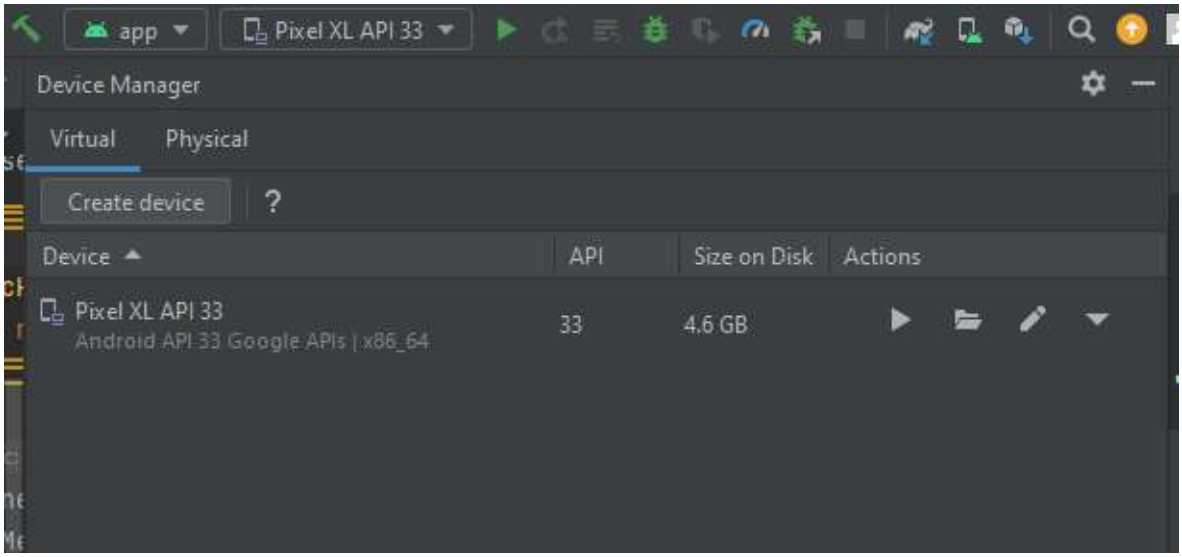


Joonis 5.2 Android Studio projekti seadistus

Järgnevalt peab kasutaja valima mobiilirakenduse käivitamiseks sobiva tööriista. Rakendada saab Android nutiseadet või emulaatorit. Android Studio arenduskeskkonda sisseehitatud emulaator simuleerib Android operatsioonisüsteemil baseeruvaid seadmeid. Samuti võimaldab testida loodavaid rakendusi mitmel platvormil, näiteks tahvelarvutides, Android TV seadmetes ning nutitelefonides. [14]

Androidi ökosüsteem on erinevate seadmete tõttu killustatud. Luues rakendusi tahvelarvutitele, nutitelefonidele ning -kelladele, on kulukas soetada iga seade eraldi testimiseks. Seejuures on Android Studio emulaatori suurim eelis testida rakendust, kui konkreetset nutiseadet ei ole selleks võimalik kasutada. Samuti pakub emulaator suurt täpsust, mis võimaldab simuleerida võrgukiiruseid, nutiseadme pööramist ning sissetulevaid kõnesid. Lisaks suudab see määrata seadme asukohta. Emulaatoris testimisel on eelis ka kiiruses, sest mobiilirakenduse andmete saatmine toimub kiiremini kui USB kaabliga ühendatud füüsilisse seadmesse. [14]

Android Studio emulaatori kasutamisel peab veenduma, et arvuti vastaks süsteeminõuetele. Mobiilirakenduse testimiseks emulaatoris peab arvutil olema vähemalt 16 GB operatiivmälu, 16 GB põhimälu ning 64-bitine *Windows, macOS, Linux* või *ChromeOS* operatsioonisüsteem. Seejärel peab kasutaja looma Android virtuaalse seadme, mis vastab loodava rakenduse nõuetele. Tööriistaribal tuleb klikkida *Device Manager*'ile ning valida sealt *Create Device* (Joonis 5.3). Järgnevalt avanevad valikud, kus on võimalik valida ekraanisuurus, resolutsioon ja Androidi ning API versioon. Järgmisena installeeritakse vajalikud failid. Seejärel on võimalik käivitada rakendus virtuaalses seadmes ning seal navigeerida. [14]



Joonis 5.3 Android virtuaalse seadme loomine

Käesoleva lõputöö autor valis töö käigus valmiva rakenduse testimiseks seadme Sony Xperia XZ F8331. Valiku põhjus tulenes rakenduse arendamiseks kasutatavast arvutist, mis ei vasta Android emulaatori miinimumnõuetele. Seetõttu oleks arendusprotsess ning testimine muutunud keeruliseks. Enne mobiilirakenduse testimist on vaja nutiseadmes luua selleks vastav võimekus ning installeerida USB draiver, mis toetab nutiseadme ühendumist Android Studio keskkonnaga. Seejärel valis autor menüüst *Settings* → *About phone* → *Software information*. Järgnevalt peab klikkima seitse korda valikule *Build number* ja sisestama seadme parooli, et avada menüü *Developer options* ning valida *Developer options on*. Pärast seda võib nutiseadme ühendada USB kaabli kaudu arvutiga ning oodata, millal Android Studio arenduskeskkond tuvastab vastava seadme. Järgmisena on võimalik kasutajal käivitada mobiilirakendus, mis alustab selle ehitust ning installeerimist nutiseadmesse.

Android Studio arenduskeskkond võimaldab ka WiFi teel nutiseadme ühendamist programmiga. Selline võimalus avaneb seadmetel, mille Android versioon on 11 või kõrgem. Ühenduse loomiseks peab kasutaja veenduma, et nutiseade ning arvuti on ühendatud samasse traadita võrku. Seejärel on vajalik värskendada Android Studio ning nutiseadme operatsioonisüsteemi versiooni. Järgnevalt tuleb avada seadmes eelpool käsitletud *Developer options* ning valida *Wireless debugging* vahelehelt *Pair device with QR code*. Android Studio tööriistaribalt tuleb valida *Pair Devices Using WiFi* ja avanenud aknast skannida QR-kood. Seejärel on võimalik testida mobiilirakendust. [15]

6. SERVERID ARENDUSPROTSESSIS

Käesolevas peatükis annab lõputöö autor ülevaate Android mobiilirakenduse arendusprotsessis kasutatud serveritest. Lisaks käsitletakse nende ülesseadmist *Windows* operatsioonisüsteemil baseeruvates seadmetes ning PostgreSQL andmebaasile kasutaja ja parooli lisamist Dockeri kaudu.

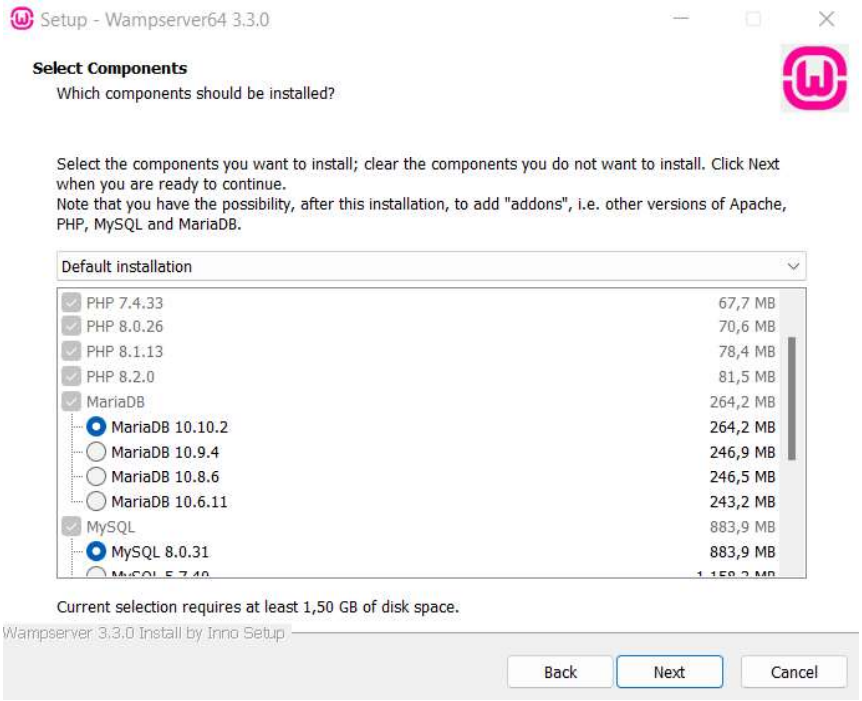
6.1 WAMP server

WAMP server on *Windows Apache MySQL PHP* tarkvara, mis võimaldab majutada veebilehti ning -rakendusi lokaalses võrgus. Lokaalne veebimajutus tähendab tagasisidestusaadressi kaudu võrguühenduse loomist ning võimaldab testida tarkvaralisi lahendusi informatsiooni saatmiseta üle võrgu. Selline tarkvaraline lahendus võimaldab kasutajal mugavalt arendada ning korrastada veebitarkvara kitsaskohti enne selle väljalaskmist. [16]

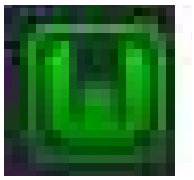
Käesolevas lõputöös kasutab autor mobiilirakenduse arendamiseks WAMP serverit. Selline valik tuleneb sellest, et oluline on hoida arendusprotsessid eraldi Eluslaboratooriumi infoekraani serverist. Andmepäringute testimiseks rakendab autor sisekliima ning ruumide nimekirja testandmeid, mis asuvad JSON formaadis WAMP testserveris. Serveri seadistamise eel on vajalik arenduseks kasutatavasse arvutisse paigaldada WAMP serveri tarkvara. Allalaadida on see võimalik leheküljelt https://www.wampserver.com/en/download-wampserver-64bits/.

Käivitades allalaetud faili peab kasutaja valima serveri kasutuskeele ning nõustuma litsentsi tingimustega. Seejärel tuleb valida, millisesse kausta paigaldada WAMP, vaikimisi pakub seade selleks „C:\wamp64". Järgmisena kuvab programm komponentide valikud (Joonis 6.1), autor soovitab valida PHP ning MySQL-i viimase versiooni. Soovitud valikute klikkimise järgselt, pakutakse WAMP serveri kausta lisamist Start-menüüsse ning alustatakse installeerimist.

Installeerimise järgselt peab kasutaja tegema valiku, millist veebibrauserit soovib vaikimisi kasutada. Käesoleva lõputöö autor valis selleks *Google Chrome* brauseri. Lokaalses võrgus töötavast WAMP-serverist annab märku roheline ikoon (Joonis 6.2), mis asub *Windows*-i peidetud ikoonide jaotises.



Joonis 6.1 WAMP serveri pakutavad komponendid



Joonis 6.2 Aktiivses olekus WAMP-serveri ikoon

Järgmisena peab kasutaja suunduma WAMP serveri avalehele (Joonis 6.3) sisestades veebilehitseja aadressiribale IP-aadressi. Seejärel loob kasutaja veebilehe projekti jaoks kausta „C:\wamp64\www" kataloogi ning lisab *virtual host*-i, mis võimaldab veebiserveril majutada domeene ning veebirakendusi. Seejärel täidetakse andmeväljad veebilehe loomiseks, lõputöö autor andis *virtual host*-ile nime *hostName*. Järgnevalt tuleb käivitada veebimajutuse loomine ning restartida DNS. Suundudes tagasi leheküljele *localhost*, on alajaotuses *Your VirtualHost* tekkinud varasemalt loodud *hostName* (Joonis 6.4). Sellele peale klikkides avaneb tühi leht, sest eelpool käsitletud projektikaustas puuduvad failid ning seetõttu ei ole serveril midagi kuvada. Vajalikud failid (ruumide nimekiri ning sisekliima andmed) kloonis lõputöö autor GitLab versioonihaldusplatvormilt. Lokaalsest võrgust serverile ligipääsemiseks peab kasutaja lisama httpd-vhosts.conf faili koodilõigu (Joonis 6.5) ning muutma *Windows* operatsioonisüsteemi tulemüüri sätteid. Pärast seda restartitakse WAMP server ning kasutajal on võimalus pöörduda serveri poole arvuti sisevõrgu IP-aadressiga. Töö autor

juhib tähelepanu, et serverile ligipääsemiseks tuleb aadress sisestada kujul http://192.168.1.169/hostName. IP-aadressile järgnev osa sõltub sellest, millise nime andis kasutaja *virtual host*-ile.

Joonis 6.3 WAMP-serveri avaleht

Joonis 6.4 *Your VirtualHost* alajaotus

```
# Virtual Hosts
#
<VirtualHost *:80>
  ServerName localhost
  ServerAlias localhost
  DocumentRoot "${INSTALL_DIR}/www"
  <Directory "${INSTALL_DIR}/www/">
    Options +Indexes +Includes +FollowSymLinks +MultiViews
    Require all granted
  </Directory>
</VirtualHost>

#
<VirtualHost *:80>
        ServerName tempinfo
        DocumentRoot "c:/wamp64/www/tempinfo"
        <Directory  "c:/wamp64/www/tempinfo/">
                Options +Indexes +Includes +FollowSymLinks +MultiViews
                Require local
        </Directory>
</VirtualHost>
```

Joonis 6.5 Faili httpd-vhosts.conf sisu

6.2 Docker platvorm

Docker on platvorm, mis kasutab klient-teenindaja arhitektuuri ning koosneb kolmest komponendist: REST API, server ning käsuviip. Edasi jaguneb see neljaks: Dockeri server, *Containerd*, *Containerd-shm* ning *RunC*. Dockeri server on pidevalt töötav programm, mis haldab ja kontrollib konteinereid, imidžeid (konteineri versioon, mis sisaldab tarkvara ning parameetreid rakenduse käivitamiseks) ning veebivõrke. Konteinerite elutsükli haldamiseks kasutatakse *Containerd* tööriista, mis rakendab omakorda *RunC*-d konteineri käivitamiseks. Käsitletav käsurea tööriist loob konteineri kasutades nimesalve, kontrollrühmi, failisüsteemi juurdepääsu kontrolli ning *Linux* turvasüsteeme. Konteineri käivitamise järgselt annab *RunC* haldamise edasi *Containerd-shim* liidesele. [17]

Lõputöö käigus valmiv rakendus hakkab kasutama Eluslaboratooriumi infoekraani kasutajaliidese API-t. Seejuures on oluline arendusprotsessi lõpus üle minna korrektsele arhitektuurile. Infoekraani kasutajaliides on pakitud Dockeri konteineritesse. Töö autor kasutab arvutis veebiserveri ning infoekraani käivitamiseks Dockeri platvormi. Järgnevalt käsitletakse platvormi seadistamist *Windows* operatsioonisüsteemiga seadmes.

Docker Desktop on rakendus, mis käivitab Dockeri konteinerid *Windows* operatsioonisüsteemis. Enne desktop platvormi paigaldamist on seadmesse vaja installeerida WSL (*Windows Subsystem for Linux*). Selleks peab kasutaja avama

administraator õigustega terminali ning sisestama käsu wsl --install. Seejärel on vaja arvuti taaskäivitada ja avanenud konsooli sisestada kasutajanimi ning parool. Väljumisel *Linux*'i terminalist peab kasutaja sisestama käskluse *exit*. Uuesti saab seda käivitada sisestades terminali käskluse wsl --user <kasutajanimi>. Järgnevalt on vaja allalaadida Docker Desktop installer leheküljelt: https://www.docker.com/products/docker-desktop/. Allalaadimise järgselt tuleb konfiguratsiooni aknas valida *Use WSL 2 instead of Hyper-V* ning arvuti taaskäivitada. Seejärel peab kasutaja nõustuma Dockeri teenuse tingimustega ning uuendama WSL-i sisestades administaator õigustega terminalis käskluse *wsl --update*.

Enne infoekraani kasutajaliidese failide kloonimist, peab liikuma terminalis kausta, kuhu soovitakse repositoorium paigaldada ning sisestama käskluse *git config --global core.autocrlf input*. Vastav käsklus tagab selle, et kloonimise käigus ei toimuks reavahetused *Linux*'i platvormilt *Windows*-ile. Seetõttu ei leia Docker lokaalsest masinast enam faile ning suunab *backend* konteineri taaskäivitus tsüklisse. Seejärel võib sisestada terminali käskluse *git clone https://gitlab.cs.ttu.ee/tartu-kolledz/eluslabor/infoscreen/infoscreen/infoscreen-docker.git*, mis kloonib Tartu kolledži infoekraani repositooriumi kasutaja seadmesse. Järgnevalt on oluline lisada arvuti IP-aadress faili .env-backend-dev või .env-backend-prod ALLOWED_HOSTS reale, et Android mobiilirakendus saaks pöörduda infoekraani API-i poole.

Infoekraani programmikoodi käivitamiseks peab kasutaja liikuma terminalis kausta, kus asub docker-compose-prod.yml või docker-composedev.yml fail, mis defineerib, milliseid konteinereid tuleb installida. Vastavas kaustas peab kasutaja sisestama käskluse *docker compose -f docker-compose-dev.yml up --build -d*. Seejärel alustatakse Dockeri konteinerite paigaldamist. Installeerimise järgselt saab kasutaja sisestada veebilehitseja aadressiribale http://localhost ning avada infoekraani avalehekülje. Dockeri konteinerite tööd saab kontrollida käsklusega *docker ps* või avada Docker Desktop platvorm, kus kuvatakse infot nende oleku kohta (Joonis 6.6).

Joonis 6.6 Dockeri konteinerite olek Docker Desktop platvormil

6.3 PostgreSQL andmebaasi seadistamine

Käesoleva lõputöö käigus valmiva mobiilirakenduse juurde kuulub tulevikus kasutajakonto ning sisselogimise süsteemi loomise võimekus. Sellega seonduvalt kasutab töö autor Dockeri platvormi, et luua adminkasutaja PostgreSQL andmebaasi haldamiseks. Andmebaasi adminkasutaja loomiseks tuleb liikuda terminalis kausta, kus asuvad infoekraani rakenduse failid. Seejärel tuleb sisestada järgnev käsklus: *docker compose -f docker-compose-prod.yml exec backend python manage.py createsuperuser*. Järgnevalt peab kasutajale andma nime ja trükkima Emaili aadressi ning parooli. Seejärel edastab terminal kasutajale järgmise teate: *Superuser created successfully*. Andmebaasi sisenemiseks tuleb veebilehitseja aadressiribale sisestada *http://localhost/admin*, kus küsitakse kasutajanime ning parooli.

7. RAKENDUS

Käesolevas peatükis annab autor ülevaate lõputöö käigus valminud mobiilirakenduse arendusprotsessist. Seejuures käsitleb graafiliseliidese disainimist, koodi arhitektuuri, esile kerkinud probleeme ning lahendusi. Samuti kirjeldab töö autor rakenduse tegevuste tsükleid.

7.1 Objekti ülevaade

Lõputöö käigus valminud Android mobiilirakendus on arendatud kahes peamises etapis. Töö esimeses osas disainis autor rakenduse leheküljed. Seejärel projekteeris draw.io keskkonnas vooskeemi (Joonis 7.1), mis kirjeldab, missugust ülesannet mingis etapis lahendatakse. Projekti teises osas alustas autor tarkvaraarendust võttes aluseks varasemalt loodud skeemi.

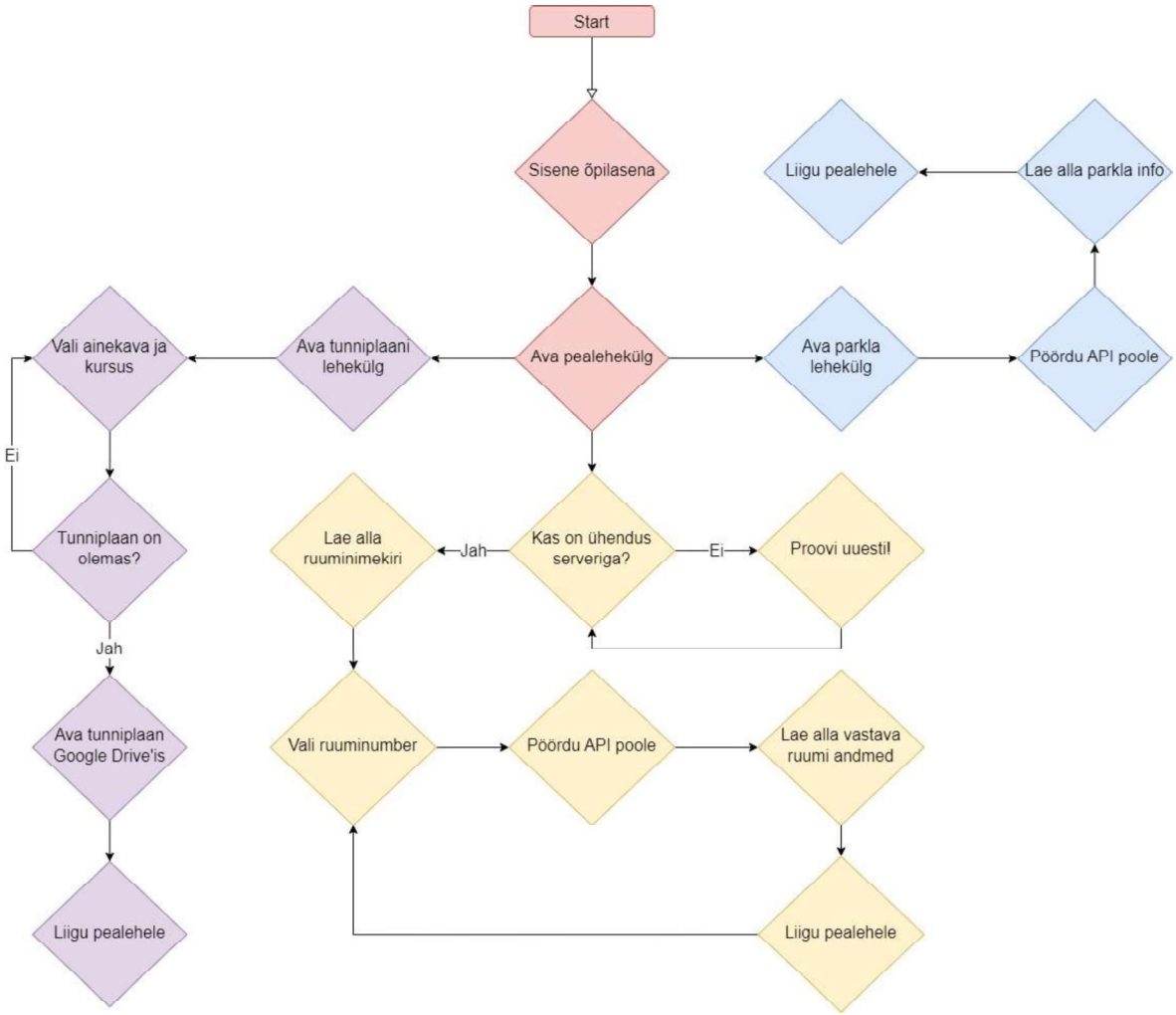
Käesoleva rakenduse voo kirjeldamisel on autor lähtunud eelkõige tavapärasest kasutamisest. Android mobiilirakenduse esmasel käivitusel kuvatakse kasutajale sisselogimise lehekülge. Sealt edasi on võimalik suunduda ka konto loomise lehele. Lõputöö skoobist ning mahust lähtudes on autor jätnud mõlema lehekülje *Backend*'i arendamise tööst välja ning keskendunud külalisvaate loomisele. Sisenedes õpilasena põhivaatesse, ei nõuta kasutajatunnuse ning parooli sisestamist.

Pealeheküljele jõudes kuvatakse kasutajale alamenüü, mis jaguneb kolmeks: kodu, tunniplaan ning parkla. Järgmisena teostatakse päring, mis kontrollib serveri kättesaadavust. Juhul kui ühendus puudub, korratakse protsessi. Tsükkel kestab seni kuni luuakse internetiühendus mobiilirakenduse ning serveri vahel. Seejärel pöördutakse HTTP protokolli alusel GET-tüüpi päringuga InfoAPI poole, et allalaadida Eluslaboratooriumi ruumide nimekiri. Järgnevalt saab kasutaja valida, millise ruumi sisekliima andmeid soovib vaadelda. Sarnaselt nimekirja allalaadimisega, edastab rakendus GET päringu InfoAPI-le võttes argumendiks konkreetse ruumi numbri. Kui päring on õnnestunud, kuvatakse kasutajale sisekliima andmete vaheleheküljel temperatuuri, suhtelise õhuniiskuse ning süsihappegaasi taset. Seejärel võib liikuda tagasi pealehele.

Tunniplaani nägemiseks peab kasutaja valima menüüst vastava ikooni. Järgnevalt suunab rakendus vahelehele, kus valitakse kursus ning ainekava, mida soovitakse vaadata. Seejärel on võimalik avada tunniplaan *Google Drive* keskkonnas klikkides nupule *Näita tunniplaani*. Juhul kui vastavat linki ei ole rakenduses defineeritud, kuvab

rakendus kasutajale teate *Selline tunniplaan puudub*. Kirjeldatud protsessi korratakse seni kuni leitakse olemasolev tunniplaan. Pärast seda võib liikuda tagasi põhileheküljele.

Tulevikus on oluline, et lõputöö rakendus suudab parklas olevate sõidukite hulka kuvada. Funktsionaalsuse arendamisel on autor kasutanud testandmeid. Info nägemiseks peab kasutaja valima alamenüüst parkla vahelehekülje. Seejärel teostab rakendus GET-päringu InfoAPI suunas. Kui päring õnnestus, laetakse alla sõidukite arvu käsitlev info. Järgnevalt võib suunduda tagasi mobiilirakenduse pealehele.

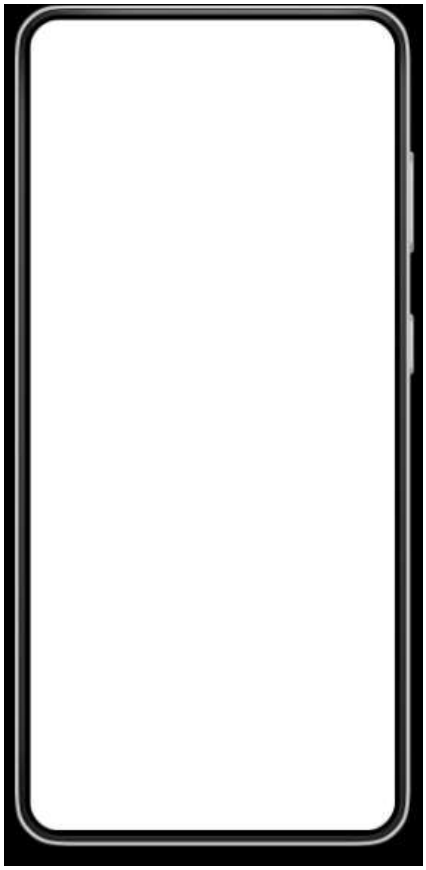


Joonis 7.1 Lõputöö rakenduse vooskeem

7.2 Graafiliseliidese disain

Lõputöö käigus valminud Android mobiilirakenduse kujundamisel kasutas autor Figma disani keskkonda. Disainimisel on eelkõige lähtutud sellest, et lõputöö Esimeses etapis disainiti kasutajasõbralik kasutajaliides. Selle hulka kuulusid järgnevad vaheleheküljed:

kasutaja loomine, sisselogimine, ruumide nimekiri, sisekliima andmed, tunniplaan ning parkla info. Esmalt valis autor Figma keskkonnas prototüübi loomiseks Android nutiseadme põhja mõõtmetega 360 x 800 pikslit (Joonis 7.2). Seejärel oli ülesanne kujundada sobiv taust, mis iseloomustaks seda Tartu kolledžis kasutatava rakendusena. Järgmisena paigutas autor vajalikud disaini elemendid nutiseadme ekraanile ja valis sobivad värvid ning suurused.

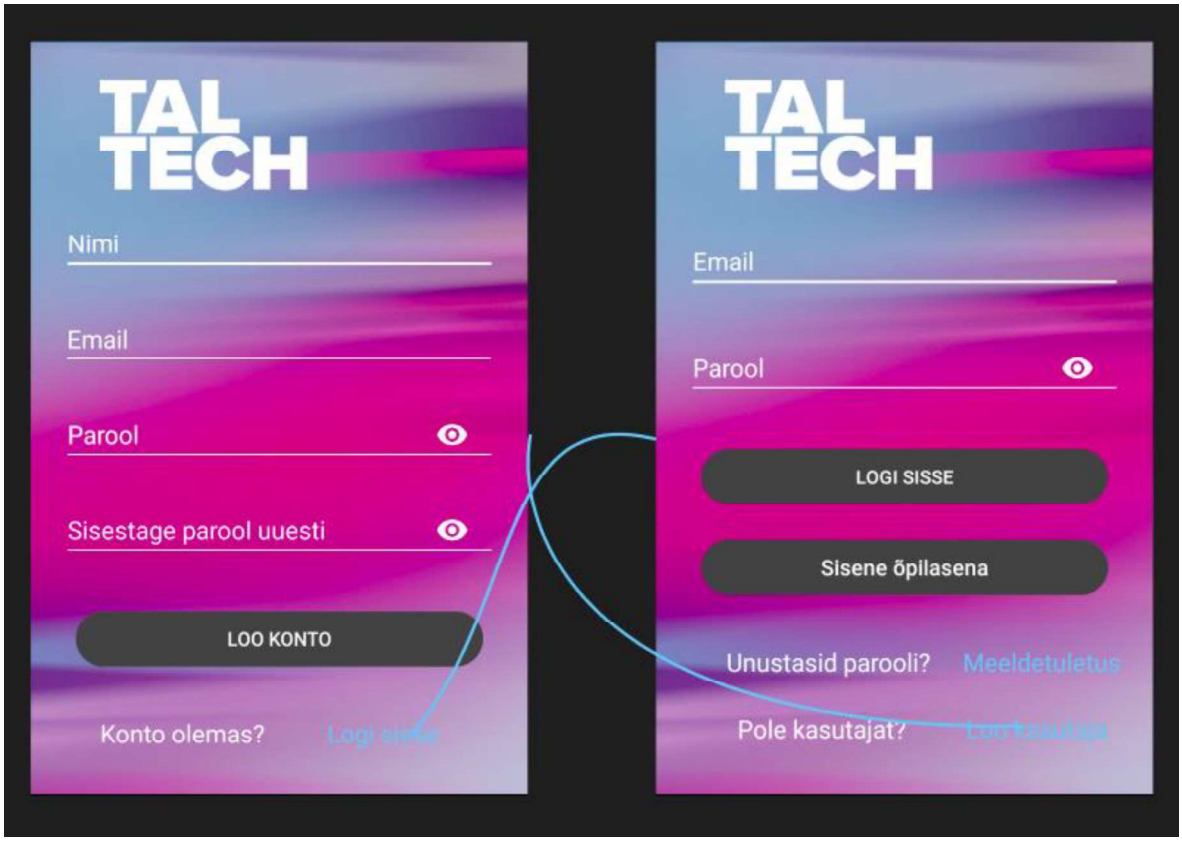


Joonis 7.2 Disaini prototüübi põhi

Android mobiilirakenduse sisselogimise leheküljele paigutas autor emaili ja parooli sisestamise jaoks vajalikud lahtrid. Seejärel lisati interaktsioonide jaoks nupud *LOGI SISSE* ning *Sisene õpilasena*. Tekstinupp *Loo kasutaja* viib kasutaja konto loomise leheküljele.

Kasutajakonto loomise leheküljel on kujutatud järgnevad neli lahtrit: Nimi, Email, Parool ning Sisestage parool uuesti. Seejuures kasutas autor sarnaseid disaini elemente eelmise lehega. Seejärel lisas lõputöö autor nupu *LOO KONTO*, mis võimaldab tulevikus käivitada konto loomise protsessi. Tekstinupp *Logi sisse* viib kasutaja tagasi sisselogimise lehele.

Figma disaini keskkond pakub kasutajale voogude projekteerimise võimalust. Selliselt on võimalik arendada rakenduse kasutajakogemus ehk *User experience*. Antud keskkonnas kujundas lõputöö autor lehekülgede vahetuse voo (Joonis 7.3) järgnevalt: kui kasutaja klikib tekstile *Loo kasutaja*, suunatakse kasutaja konto loomise lehele ning kui klikitakse tekstile *Logi sisse*, liigutakse tagasi sisselogimise leheküljele. Figma keskkonnas arendas autor ka teiste vahelehtede vahel liikumise loogika.



Joonis 7.3 Kasutajavoog sisselogimise ning kasutaja loomise lehekülgede vahel

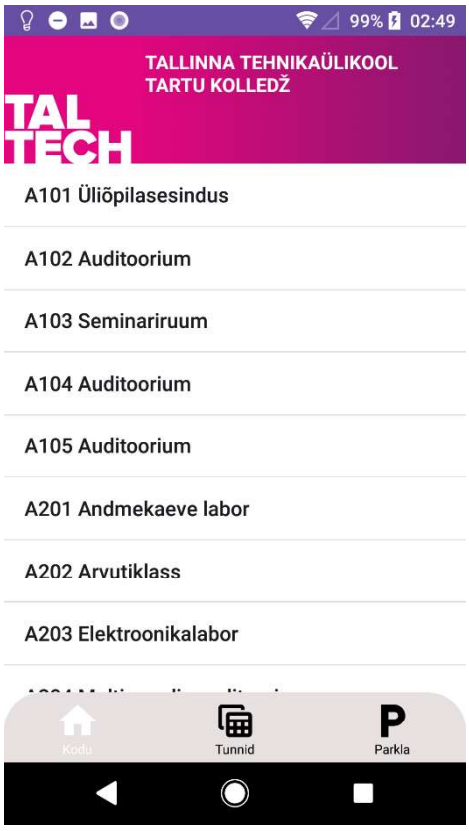
Järgnevalt kujundas autor ruumide nimekirja ning sisekliima andmete kuvamise vaheleheküljed. Mobiilirakenduse päises on kujutatud TalTechi logo ning sellele omaseid värve. Alloleval joonisel on kujutatud ruumide nimekirja peale allalaadimist (Joonis 7.4). Juhul kui serveriga ühendus puudub kuvab rakendus kasutajale ooterežiimi (Joonis 7.5). Käsitletud funktsionaalsus on arendatud hilisemas tarkvaraarenduse etapis.

Sisekliima andmete kuvamise lehekülg on disainitud sarnaselt põhileheküljega. Antud vaheleht omab päist, kus on esile toodud eelpool mainitud disaini elemendid. Lisaks on leheküljele paigutatud andmeväljad, kus kuvatakse ruumi numbrit, temperatuuri, süsihappegaasi taset ning suhtelise õhuniiskuse hulka (Joonis 7.6). Vaikimisi on võrgu ühenduse puudumise korral sisekliima andmete väärtused kujutatud sümboliga "-".

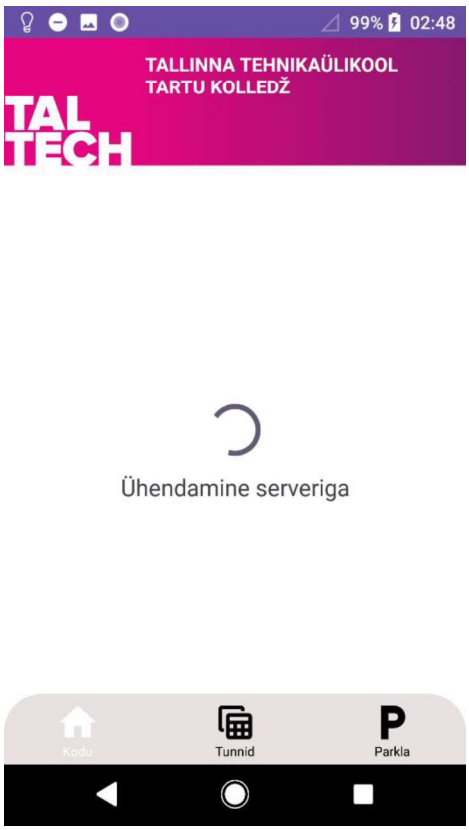
Tunniplaani kuvamise vahelehel (Joonis 7.7) on kujundamisel lähtutud samuti varasemalt disainitud päise kasutamisest. Seejuures on lisatud nupp *Näita tunniplaani*, mille funktsionaalsus on arendatud lõputöö teises etapis. Leheküljel olev kursuse numbri ning ainekava valimine on disainitud tarkvaraarenduse protsessis.

Sarnaselt sisekliima andmete vaheleheküljega kujundas autor parkla info kuvamise lehe (Joonis 7.8). Seejuures on seal rakendatud varasemalt käsitletud disaini elemente. Andmeväljas kuvatakse kasutajale parklas olevate sõidukite arvu. Võrguühenduse puudumisel on andmeväärtus vaikimisi asendatud sümboliga "-". Kõikidel vahelehekülgedel kujutatud menüü on arendatud lõputöö teises etapis.

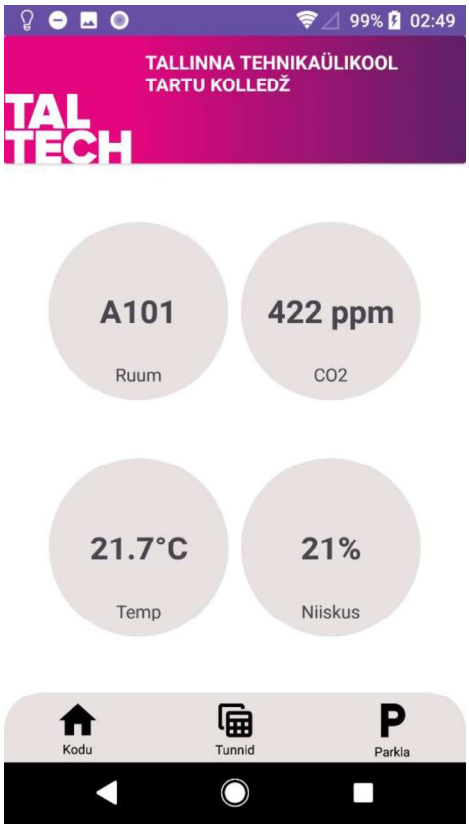
Figma disaini keskkond pakub kujunduse ümberkonverteerimise võimekust. See võimaldas töö autoril teisendada disainitud vaheleheküljed XML-märgendkeele failideks. Seejärel rakendati antud faile Android Studio arenduskeskkonnas. Selline lahendusviis muutis tööprotsessi efektiivsemaks ning aitas teha kujunduse muudatusi vastavalt vajadustele.



Joonis 7.4 Ruumide nimekirja vahelehekülg



Joonis 7.5 Serveriga ühenduse saamine



Joonis 7.6 Ruumi A101 sisekliima andmete vahekülg

Joonis 7.7 Tunniplaani vahelehekülg

Joonis 7.8 Parka info vahelehekülg

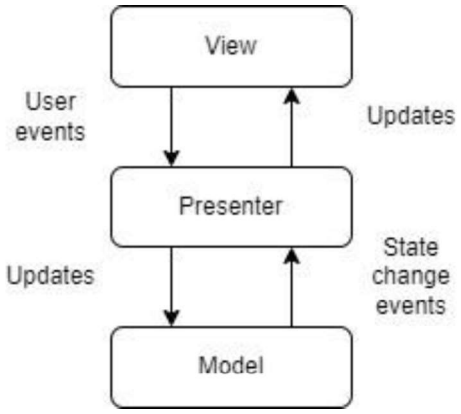
7.3 Tarkvaraarendus

Käesoleva lõputöö käigus valminud mobiilirakenduse arendamisel on kasutatud *Waterfall* tarkvaraarenduse paradigmat. Käsitletava mudeli puhul on tegemist klassikalise projektijuhtimisega. Seejuures edastab klient oodatavad tulemused projekti alguses. Samuti on fookus iga etapi selgel määratlemisel ning eesmärgipärasel teostusel. [18] Lähtudes lõputöö eesmärkidest, teostas autor iga seatud ülesande kindlas järjekorras.

Lõputöö rakenduse arendamisel on autor lähtunud *MVP* ehk *View-Model-Presenter* mustri (Joonis 7.9) kasutamisest. Käsitletav arhitektuuri mudel on laialdaselt levinud tarkvaraarenduse valdkonnas. Järgnevalt kirjeldab autor selle tööprotsessi.

Esmalt teostab kasutaja toimingu, mille *View* ehk vaade edastab *Presenter*'ile ehk esitajale. Seejärel teavitab *Presenter Model*'it ehk mudelit, mida soovitakse uuendada. Pärast andmete uuendamist saadetakse need tagasi esitajale töötlemiseks. Järgnevalt tagastatakse uuendused vaatele, mis kuvab tulemuse kasutajale. [19]

Lõputöö käigus valmiva rakenduse *View* komponendi moodustavad *Activity* ehk tegevuse klassid ning *layout* failid. Esitaja rolli täidab selles rakenduses *DataParser.kt* klass, mis loeb sisse kasutaja interaktsioonid. Vajalikud päringud edastatakse InfoAPI-le, mis suunab need edasi serverile. Lähtudes rakenduse arhitektuuri mallist, täidab lõputöö projektis mudeli rolli Infoekraani serveri andmebaas. Järgnevalt tagastatakse vastavad andmed *DataParser.kt* klassile, mis suunab need edasi *View* komponendile. Seejärel uuendatakse rakenduse vaadet ning kasutajale kuvatakse soovitud andmed.



Joonis 7.9 *MVP* arhitektuuri mudel

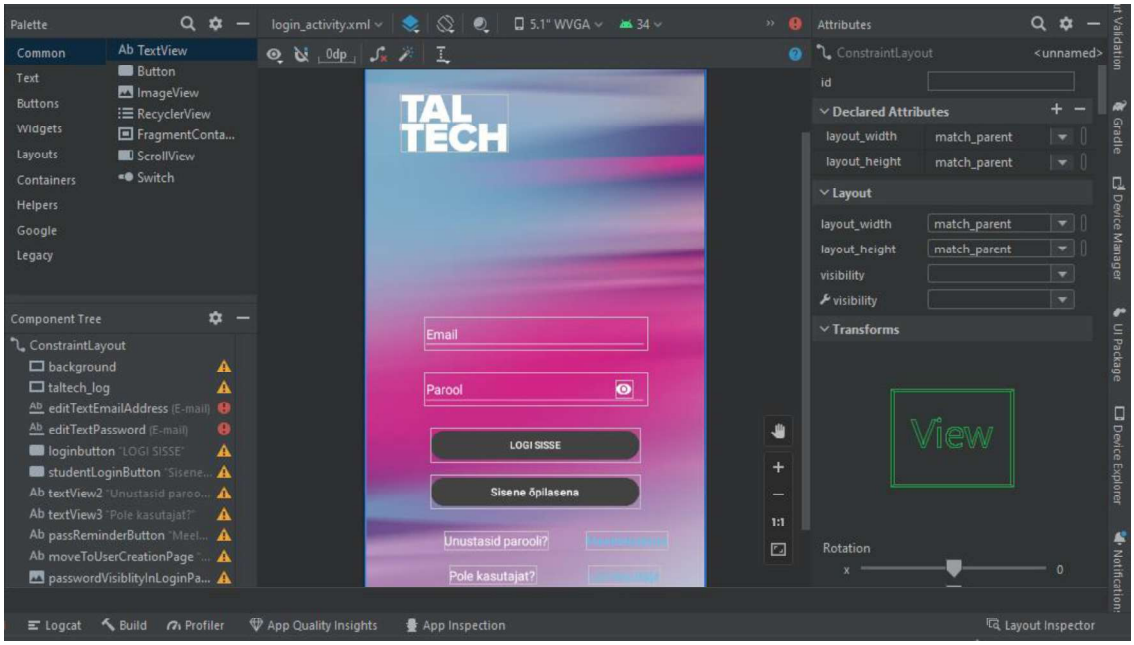
7.3.1 Manifest

Android Studio failis AndroidManifest.xml hoiustatakse mobiilirakenduse teavet. Sealhulgas on kirjeldatud rakendusele antavad õigused, nimi, ikoon ning kasutusel olevad *Activity*'d ehk tegevused. [20] Lõputöö autor on seal defineerinud internetiühendusele antavad nõusolekud ning järgmised *Activity*'d ehk tegevused: *LoginActivity*, *CreateUserActivity*, *MainActivity*, *RoomDataActivity*, *DataParser*, *ActivityChanger*, *LessonPlanActivity* ning *ParkingActivity*.

7.3.2 Kasutajaliides

Android Studio arenduskeskkonnas asuvas ressursside kaustas hoiustatakse kasutajaliidesele olulisi komponente, sealhulgas ekraani vahelehekülgede kujundus, pildid ning kirjastiilid. Samuti asub kaustas alamkaust väärtuste jaoks, kus deklareeritakse kirjasuurused, värvitoonid ning nende tugevused. [20] Lõputöö autor on kaustas deklareerinud kuus peamist *layout* faili: *activity_main.xml*, *data_activity.xml*, *login_activity.xml*, *signup_activity.xml*, *parking_activity.xml* ning *lesson_plan_activity.xml*. Antud failides asuvad XML-märgendkeelt kasutavad vaheleheküljed.

Lisaks koodi kirjutamisele, võimaldab Android Studio mobiilirakenduse visuaalset kujundamist sarnaselt Figmaga. See pakub laia valikut eritüüpi nuppudest ning vaadetest (Joonis 7.10), mis võimaldas lõputöö autoril teha soovitud muudatusi otse keskkonnas (näiteks alamenüü loomine).



Joonis 7.10 Mobiilirakenduse disainimine Android Studio keskkonnas

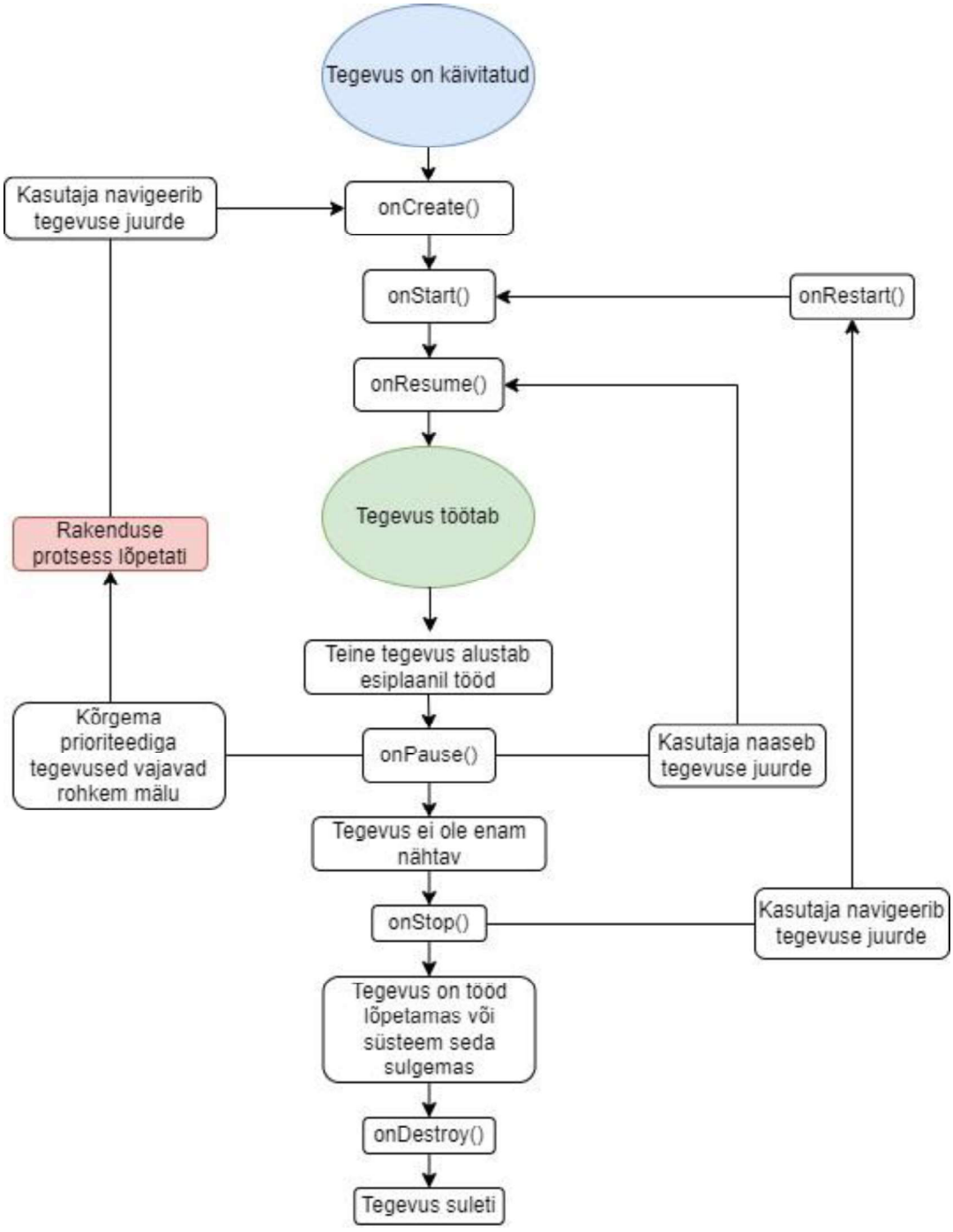
7.3.3 Koodi arhitektuur

Lõputöö käigus valminud rakenduse programmeerimisel on kasutatud *DRY* ehk *Don't Repeat Yourself* printsiipi. Sellega seonduvalt lähtus autor koodi kirjutamisel sellest, et luua võimalikult palju taaskasutatavaid komponente.

Android mobiilirakendus kasutab tavaliselt rohkem kui ühte tegevust, on oluline nende toimimist hallata. Tegevuse elutsükli haldusega tegeleb mobiilirakenduses pinu. Käivitades tegevuse suunatakse see süsteemi pealmiseks elemendiks ja teised varasemalt töös olnud tegevused liiguvad madalamale positsioonile ning neid ei kuvata enam ekraanil. Mobiilirakenduse tegevusel on neli seisundit:

- Ekraani peal toimuv on kõige pealmine pinu tegevus.
- Tegevus, mis on seatud pausile, on kasutajale nähtav, kuid ei ole fookuses. Lisaks võib niisuguse tegevuse süsteem automaatselt sulgeda, kui rakendusel on vaba mälu ressursid piiratud.
- Tegevus on kaetud teise tegevusega ning süsteem on peatanud selle töö.
- Tegevus on peatatud ja süsteem võib kõik protsessid sulgeda ning vabastada mälu. Niisuguses seisundis tegevuse uuesti kasutamisel peab sooritama restardi.

Mistahes tegevuse puhul läbitakse selle elutsükli jooksul kõiki eelpool käsitletud seisundeid. [20] Rakenduse elutsüklit selgitab alljärgnev joonis (Joonis 7.11).



Joonis 7.11 Rakenduse elutsükkel

Lõputöö rakenduse *Backend*'i moodustavad järgmised Kotlini klassid: *ActivityChanger.kt*, *CreateUserActivity.kt*, *DataParser.kt*, *LessonPlanActivity.kt*, *LoginActivity.kt*, *MainActivity.kt*, *ParkingActivity.kt* ning *RoomDataActivity.kt*. Mobiilirakenduse avamisel sisselogimise vaate loomisel kutsutakse klassis *LoginActivity.kt* välja *onCreate ()* meetod, kus deklareeritakse kasutatavad nupud (LOGI SISSE, Sisene õpilasena ning Loo kasutaja) ning tekstilahtrid. Samuti kutsutakse seal

välja meetod *togglePasswordVisibility()*, mis muudab lahtrisse sisestatud parooli kasutajale nähtavaks ning peidetuks.

Liikudes kasutaja konto loomise vaatesse, kutsutakse välja klass *CreateUserActivity.kt*. Klassis käivitatakse meetod *onCreate()*, milles on ära deklareeritud neli tekstilahtrit (Nimi, Email, Parool ning Sisestage parool uuesti) ning nupud (LOO KONTO ning Logi sisse). Sarnaselt eelnevalt käsitletud klassiga, kutsutakse seal välja meetod *togglePasswordVisibility()*, mis muudab parooli nähtavust.

Põhilehekülje vaate loomisel kutsutakse välja klass *MainActivity.kt*. Seal käivitatakse meetod *onCreate()*, milles kutsutakse välja klassid *DataParser.kt*, *ActivityChanger.kt* ning meetod *checkServerConnection()*. Klassis *MainActivity.kt* alustab tööd *checkServerConnection()*, mis kutsub välja meetodi *isServerAccessible()*. Seal kontrollitakse internetiühendust veebiserveriga. Tsüklit korratakse seni kuni luuakse ühendus serveri ning mobiilirakenduse vahel. Seejärel kutsub *checkServerConnection()* välja meetodi *fetchRoomNrList()*. Käsitletavas meetodis kutsutakse välja klass *DataParser.kt*. Klassis on deklareeritud järgmised meetodid: *fetchRoomNrList()*, *fetchRoomDataList()* ning *fetchCarParkData()*. Meetodis *fetchRoomNrList()* teostatakse GET-tüüpi ruumide nimekirja päring API suunas. Seejärel laetakse alla *JSON*-massiiv, mille iga element muudetakse *JSON*-objektiks. Objektide hulgast saadakse kätte muutujad *roomNr* ning *description* ning lisatakse *roomNrList* massiivi. Meetodis *fetchRoomNrList()* kutsutakse välja alammeetod *onRoomNrListFetched()*, milles lisatakse ruumide nimekiri *ArrayAdapter*'i abil loendivaatesse ehk *ListView*'sse.

Järgnevalt kirjeldab lõputöö autor ruumide sisekliima andmete allalaadimist serverist. Kui kasutaja on mobiilirakenduses klikkinud ruuminumbrile, loetakse see argumendina sisse ning alustatakse sisekliima andmete lehekülje vaate loomist. Seejuures kutsutakse välja klass *RoomDataActivity.kt*, milles on deklareeritud järgmised meetodid: *onCreate()*, *ActivityChanger()* ning *fetchRoomDataList()*. Meetodis *onCreate()* käivitatakse *fetchRoomDataList()*, mis kasutab *DataParser.kt* klassi andmete allalaadimiseks. Käsitletavas klassis kutsutakse välja meetod *fetchRoomDataList()*. Sarnaselt ruumide nimekirja allalaadimisega, pöördutakse GET-tüüpi päringuga API poole. Seejärel laetakse alla *JSON*-massiiv, mis teisendatakse *JSON*-objektideks. Objektide hulgast võetakse välja muutujad temperatuur, süsihappegaas, õhu suhteline niiskus ning ruuminumber, mis lisatakse *roomDataList* massiivi. Klassis *RoomDataActivity.kt* välja kutsutud meetodis *fetchRoomDataList()* omistatakse massiivi väärtused *onCreate()* funktsioonis deklareeritud muutujatele.

Lõputöö autor annab ülevaate parkla info allalaadimise ning kuvamise metoodikast. Tartu kolledži olevate sõidukite arvu kuvamiseks peab kasutaja klikkima alamenüüs

parkla ikoonile. Seejärel alustatakse vahelehekülje vaate loomist, milles kutsutakse välja klass *ParkingActivity.kt*. Käsitletavas klassis on deklareeritud meetodid *onCreate()* ning *fetchCarParkData()*. Järgnevalt käivitatakse koos *fetchCarParkData()* meetodiga klass *DataParser.kt*. Klassis *DataParser.kt* asuvas *fetchCarParkData()* meetodis andmete allalaadimisel teostatakse GET-päring API suunas ning laetakse alla *JSON*-objekt. Objektist võetakse välja sõne *cars* ning selle arvuline väärtus omistatakse klassis *ParkingActivity.kt* asuvas meetodis *onCarParkDataFetched()* muutujale *amountOfCars*.

Tunniplaani nägemiseks peab kasutaja klikkima menüüs vastavale ikoonile. Seejärel alustatakse vahelehekülje vaate loomist kutsudes välja klassi *LessonPlanActivity.kt*. Klassis on deklareeritud iga Tartu kolledži ainekava kursuse tunniplaan ning meetodid *onCreate()*, *openBrowserLink()* ning *updateSelectedStringElement()*. Meetodis *updateSelectedStringElement()* toimub ainekava ning kursuse numbri valimine. Seejärel liidetakse kaks sõne osa ühtseks tervikuks. Klikkides nupule *Näita tunniplaani*, kontrollitakse *onCreate()* meetodis, kas vastav tunniplaan on olemas. Juhul kui tunniplaan puudub peab kasutaja uuesti valima mõne teise ainekava või kursuse numbri. Olemasoleva valiku korral kutsutakse välja meetod *openBrowserLink()*, mis avab vastava veebilingi Google Drive's.

Lõputöö rakenduses oleva alamenüü tööd teostab klass *ActivityChanger.kt*. Käsitletavas klassis on deklareeritud menüü nupud (Kodu, Tunniplaan ning Parkla). Klass koosneb meetoditest *onCreate()* ning *setFocus()*. Meetodis *setFocus()* liigutakse sellele vahelehekülgele, mille ikoonile kasutaja alamenüüs klikkis.

8. TULEMUSTE ANALÜÜS

Käesolevas peatükis annab autor ülevaate lõputöö tulemustest. Lisaks toob esile lõputöö käigus ilmnenud takistused, nende lahendused ning mida oleks teinud teisiti. Samuti analüüsib, milliseid väärtusi omab valminud mobiilirakendus Tartu kolledži õppeprotsessis. Seejärel teeb ettepanekud tulevasteks edasiarendusteks.

Lähtuvalt lähteülesandest ning sellest tulenevatest alaülesannetest on autor täitnud kõik seatud eesmärgid. Valminud on Android mobiilirakenduse esmane versioon, mis võimaldab Tartu kolledži kollektiivil jälgida A-kopruse õpperuumide sisekliima informatsiooni. Samuti kuvab rakendus kasutajale tunniplaani vastavalt kursusele ning ainekavale. Lisaks on loodud võimekus tulevikus vaadata kolledži parklas olevate sõidukite arvu ning arendada sisselogimise ja kasutaja loomise süsteem.

Samuti omab loodud rakendus väärtust õppematerjalina. Mobiilirakendust on võimalik rakendada praktikumides analüüsides A-korpuse sisekliima parameetreid. Lisaks avaneb Tartu kolledži tudengitel võimalus tutvuda Android mobiilirakenduse arendamisega. Teostades edasiarendusi Eluslaboratooriumi projektõppe või lõputööde raames, saavad tudengid väärtusliku kogemuse tarkvaraarenduses.

Lõputöö autor juhib tähelepanu, et valminud lõputöö aitab tudengitel mõista Android mobiilirakenduse ülesehituse põhitõdesid. Samuti annab see ülevaate, kuidas arendada rakendust, mille fookus on suunatud tarkadele kodudele ning hoone sisekliima jälgimisele. Kasutades lõputöö käigus valminud materjale, saavad Tartu kolledži tudengid tutvuda, kuidas arendada rakenduse ja serveri vahelist andmesidet.

Järgnevalt toob autor esile probleemid, mis kujunesid lõputöö käigus. Esimene probleem tekkis Docker Desktop rakenduse seadistamisel. Nimelt pärast esmakordset ülesseadmist, kujunes takistuseks infoekraani rakenduse ning serveriga ühenduse saamine. Probleemi tuum seisnes selles, et lõputöö autor oli klooninud Tartu kolledži versioonihaldusplatvormilt GitLab infoekraani rakenduse repositooriumi, mis on ülesehitatud *Linux*'i platvormi peale. Seejärel oli *Windows* sooritanud reavahetused ning ei leidnud lokaalsest masinast enam üles faile, mis tekitas olukorra, kus infoekraani *backend* konteiner jäi restart tsüklisse.

Kuna lõputöö autor kasutab *Windows* operatsioonisüsteemil baseeruvat seadet, oli oluline enne repositooriumi kloonimist liikuda terminalis kausta, kuhu seda soovitakse ning seejärel sisestada käsk: git config --global core.autocrlf input. Antud käsklus tagab selle, et ei toimuks reavahetusi *Linux*'i platvormilt *Windows*-ile. Järgnevalt kloonis autor

uuesti infoekraani projekti ning uuesti Dockeri platvormi seadistamise järel konteinerite tööajal probleeme ei esinenud.

Lõputöö käigus valminud mobiilirakenduse programmeerimisel kujunes suurimaks takistuseks *AsyncTask*'i kasutamine. Nimelt kuna autor oli varasemalt tuttav just selle *Thread*'iga, rakendati seda esialgses koodi versioonis andmete allalaadimiseks. Siiski ei toeta Android Studio enam käsitletava *Thread*'i kasutamist ning andmete allalaadimiseks võeti kasutusele *Coroutine*. Sellega seonduvalt pidi autor end kurssi viima vastava teema ning sellega, kuidas rakendada seda sobivalt andmete allalaadimiseks.

Samuti oli tööprotsessi juures keeruline mobiilirakenduse koodi optimeerimine ning analüüsimine. Lõputöö käigus oli oluline lähtuda tarkvaraarenduse paradigmast *Don't Repeat Yourself*. Mistõttu oli oluline vältida ebavajalikke kordusi koodis. Lisaks oli oluline mobiilirakendus ülesehitada korrektsele arhitektuurile kasutades selleks Tartu kolledži infoekraani serverit.

Järgnevalt toob käesoleva lõputöö autor välja võimalikud ideed tulevikuks. Antud mobiilirakenduse võib üleviia *React Native* raamistikule. Selline lahendus lubab ühe koodibaasiga arendada rakenduse mitmele erinevale platvormile, näiteks iOS, Android ja veebirakendused.

Android mobiilirakendusse võib sisse ehitada veel teavituste edastamise funktsionaalsuse, näiteks, kui mõne ruumi süsihappegaasi tase tõuseb üle piirnormi. Olenevalt Tartu kolledži edaspidisest võimekusest, võib kaaluda mobiilirakendusse hoone küttesüsteemi juhtimise võimaluse programmeerimist. Samuti võib arendada andmegraafikute näitamise võimekuse, mis võimaldab jälgida, näiteks mõne päeva taguseid andmeid. Lisaks toob autor esile, et tulevikus võib arendada rakenduses ka sisselogimise ning kasutaja loomise süsteemi.

9. KOKKUVÕTE

Käesoleva lõputöö eesmärgiks oli luua Tartu kolledži Eluslaboratooriumisse Android operatsioonisüsteemil baseeruv mobiilirakendus, mis imiteerib A-korpuse seinal olevat infoekraani rakendust. Antud mobiilirakendus pidi edastama lõppkasutajale ruumide nimekirja ning sisekliima andmed. Lisaks oli oluline, et lõputöö omaks valmides väärtust õppematerjalina, mida Tartu kolledži tudengid saavad rakendada Eluslaboratooriumi projektiõppe raames. Nimetatud kursusel on õpilastel võimalik saada kogemus Android mobiilirakenduse arendamise valdkonnas ning teostada sellega seonduvalt lõputöid ja praktikume.

Android mobiilirakenduse arendamiseks kasutati Android Studio arenduskeskkonda. Serveri ja rakenduse vahelise andmeside imiteerimiseks kasutas autor arendusprotsessi esimeses pooles WAMP testserverit, mis töötas lokaalses internetivõrgus. Arenduse teises faasis liikus töö autor üle Docker testserveri kasutamisele. Mobiilirakendus programmeeriti keeles Kotlin ning vaated kujundati disaini keskkonnas Figma.

Valminud töö annab ülevaate Android mobiilirakenduse arendamisest, mis algab Android Studio keskkonna paigaldamisega seadmesse ning kirjeldab rakenduse arendamise etappe kuni valmis prototüübini. Samuti käsitletakse olemasolevaid lahendusi, mis võivad sobida lõputöö eesmärgi lahendamiseks. Järgnevate lõputööde teostamiseks, mille fookus on mobiilirakenduse arendamisel, saab antud tööd kasutada materjalina ning rakendada seda töö valmimisel.

Töö käigus sõnastas autor soovitused ning võimalused edaspidiseks mobiilirakenduse arendamiseks. Antud lõputöö käigus polnud võimalik neid puuduva tehnoloogilise ning tarkvaralise võimekuse tõttu realiseerida.

Käesoleva lõputöö teoreetilises osas kirjeldas autor lähteülesande püstitust ning sellest tulenevaid ülesandeid. Seejärel käsitles Tartu kolledži Eluslaboratooriumi tähtsust, arhitektuuri ning seadmeid sisekliima seireks. Lisaks andis autor ülevaate MeiePilve serveri ülesehitusest ja olulisusest. Samuti kirjeldas kahte mobiilirakendust sisekliima jälgimiseks. Töö praktilises osas selgitas autor arenduskeskkonna ülesseadmist ning süsteemi miinimumnõudeid Android Studio kasutamiseks. Ühtlasi kirjeldas autor rakenduse arendusprotsessi ja käsitles kasutatud tööriistu ning testservereid. Lisaks esitas töö lõpus tulemuste analüüsi.

10. SUMMARY

The aim of this thesis was to create a mobile application based on the Android operating system for the Tartu College Living Lab, simulating the information screen application on the A-building's wall. The mobile application was intended to provide the end-user with a list of rooms and indoor climate data. Additionally, it was crucial that the completed thesis would serve as valuable learning material for Tartu College students to apply in the project-based learning of the Living Lab. In this course, students have the opportunity to gain experience in Android mobile application development and carry out thesis projects and practical work related to it.

Android Studio development environment was used for developing the Android mobile application. To simulate data communication between the server and the application, the author used a WAMP test server in the first half of the development process, operating in the local network. In the second phase of development, the author transitioned to using a Docker test server. The mobile application was programmed in Kotlin, and the views were designed in the Figma design environment.

The completed work provides an overview of developing an Android mobile application, starting with the installation of the Android Studio environment on the device and describing the stages of application development until the prototype is ready. Existing solutions that may be suitable for solving the thesis's objectives are also discussed. This work can be used as material for future theses focusing on mobile application development.

Throughout the thesis, the author formulated recommendations and possibilities for further development of the mobile application. Due to the lack of technological and software capabilities, these recommendations could not be implemented in this thesis.

In the theoretical part of this thesis, the author described the formulation of the assignment and the resulting tasks. Subsequently, the importance, architecture, and devices for indoor climate monitoring at Tartu College's Living Laboratory were discussed. Additionally, the author provided an overview of the structure and significance of the MeiePilve server. Two mobile applications for indoor climate monitoring were also described. In the practical part of the thesis, the author explained the setup of the development environment and the minimum system requirements for using Android Studio. The author also described the application development process and addressed the tools and test servers used. Additionally, the thesis concluded with the presentation of the results analysis.

11. KASUTATUD ALLIKAD

[1] A. Rootsi, „Tartu Kolledž arendab laboribaasi", Tallinna Tehnikaülikool, 17.06.2021. [Võrgumaterjal]. Kättesaadav: https://taltech.ee/uudised/tartu-kolledz-arendablaboribaasi. Kasutatud 23.02.2023.

[2]. K. Jõgiste. „Ruumipõhise ventilatsioonisüsteemi juhtimine CO2 anduri sisendi kaudu Puiestee 80A loengumaja näitel", [Rakenduskõrgharidustöö]. Inseneriteaduskond, TalTech, Tartu, 2023. [Võrgumaterjal]. Kättesaadav: https://digikogu.taltech.ee/et/item/c6a74df8-974a-427f-93a7-51e9813c968b

[3] G. Randla, „EnOcean raadioliikluse jälgimise võimekuse loomine Tartu kolledži eluslaboratooriumile". [Bakalaureusetöö]. Inseneriteaduskond, TalTech, Tartu, 2020. [Võrgumaterjal]. Kättesaadav: https://digikogu.taltech.ee/en/Item/e53fbb58-82fd4a19-a40a-f0d959c88ecb

[4] S. Electric, „SE8000 Installation Guide. Replacement Procedure and Pair with ZigBee Sensors", 2020. [Võrgumaterjal]. Kättesaadav: https://www.se.com/ee/et/download/document/II-SE8000-Replacement/. Kasutatud 08.05.2023

[5] Control Products, „SCR210". [Veebileht]. Kättesaadav: https://controlproducts.com/product/scr210/#. Kasutatud 08.05.2023.

[6] M. Roosileht ja A. Rootsi, „Pilvelabori platvorm viib digiõppe uuele tasemele", Tallinna Tehnikaülikool, 03.04.2019. [Võrgumaterjal]. Kättesaadav: https://taltech.ee/uudised/pilvelabori-platvorm-viib-digioppe-uuele-tasemele. Kasutatud 28.02.2023.

[7] M. Rouse, „Mongo DB", Techopedia, 02.08.2014. [Võrgumaterjal]. Kättesaadav: https://www.techopedia.com/definition/30340/mongodb Kasutatud 03.03.2023.

[8] K. Abarenkov. „EnOcean raadioliikluse jälgimisseadme loomine Tartu kolledži eluslaboratooriumile". [Bakalaureusetöö]. Inseneriteaduskond, TalTech, Tartu, 2021. [Võrgumaterjal]. Kättesaadav: https://digikogu.taltech.ee/et/Item/249f129a-a660-4c88-bed4-91fba72b61d7

[9] Javatpoint. „JSON Example". [Võrgumaterjal]. Kättesaadav: https://www.javatpoint.com/json-example. Kasutatud 15.03.2023.

[10] uHoo, „Smart Air Monitor", 2023. [Veebileht]. Kättesaadav: https://getuhoo.com/smart-air-monitor. Kasutatud 25.03.2023.

[11] Awair, 2022. [Veebileht]. Kättesaadav: https://uk.getawair.com/. Kasutatud 25.03.2023.

[12] E. Mixon, „Android OS", TechTarget, 23.02.2023. [Võrgumaterjal]. Kättesaadav: https://www.techtarget.com/searchmobilecomputing/definition/Android-OS. Kasutatud 27.02.2023.

[13] Developers „Install Android Studio”, 2023. [Võrgumaterjal]. Kättesaadav: https://developer.android.com/studio/install. Kasutatud 27.02.2023.
[14] Developers, „Run apps on the Android Emulator”. [Võrgumaterjal]. Kättesaadav: https://developer.android.com/studio/run/emulator. Kasutatud 08.05.2023.
[15] Developer, „Run apps on a hardware device”. [Võrgumaterjal]. Kättesaadav: https://developer.android.com/studio/run/device. Kasutatud 09.05.2023.
[16] T. Dio and P. T. Akpoyibo, „Wamp Server a Technological Tool for Website Local Hosting,” *Global Scientific Journals*, vol. 9, pp. 11, lk 294-295, Nov. 2021. [Võrgumaterjal]. Kättesaadav: https://www.globalscientificjournal.com/researchpaper/WAMP_SERVER_A_TECHNOLOGICAL_TOOL_FOR_WEBSITE_LOCAL_HOSTING.pdf
[17] H. Zui and C. Gehrmann, „Lic-Sec: An enhanced AppArmor Docker security profile generator”, *Journal of Information Security and Applications*, vol. 61, pp. 2, Sept 2021. doi: 10.1016/j.jisa.2021.102924.
[18] T. Thesing, C. Feldmann and M. Burchardt, „Agile versus Waterfall Project Management: Decision Model for Selecting the Appropriate Approach to a Project,” *Procedia Computer Science*, vol. 181, pp. 746-756, 2021, doi: 10.1016/j.procs.2021.01.227.
[19] D. D. Li and X. L. Liu, „Research on MVP Design Pattern Modeling Based on MDA“, *Procedia Computer Science*, vol. 166, pp. 51-56, 2020, doi: 10.1016/j.procs.2020.02.012.
[20] A. Kaasik. „Mobiilirakendus roboti juhtimiseks“. [Bakalaureusetöö]. Arvutiteaduse instituut, Tartu Ülikool, Tartu, 2016. [Võrgumaterjal]. Kättesaadav: https://dspace.ut.ee/server/api/core/bitstreams/fe2b339a-4ca5-4704-9a4e-a08b7e043602/content