TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Anna Meier 222913IAIB
Mari Piirsalu 222984IAIB
Andreas Ayrton Luhala 223177IAIB

Täppisväetamise veebi- ja mobiilirakendus

Bakalaureusetöö

Juhendaja: Evelin Halling
Tarkvarateaduse Instituut, PhD

Tallinn 2025

Autorideklaratsioon

Kinnitame, et oleme 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.

Autorid: Anna Meier, Mari Piirsalu ja Andreas Ayrton Luhala

04.06.2025

Annotatsioon

Käesoleva bakalaureusetöö keskmes on kuus aastat tagasi JHipsteri koodigeneraatoriga loodud täppisväetamise veebirakenduse kaasajastamine ning selle kõrvale mobiilirakenduse arendamine, et mullaproovide kogumine ja analüüs oleksid sujuvamad välitöödel. Esmalt hinnati olemasoleva Angular/Spring Boot süsteemi jätkusuutlikkust. Seejärel uuendati versioonikonfliktide vältimiseks ja turvariskide maandamiseks projektis kasutatavad raamistikud viimasele versioonile ning eraldati taga- ja eesrakendus iseseisvateks mooduliteks.

Uue React Native'i mobiilirakenduse kaudu saab kasutaja skaneerida mullaproovi kotil olevat QR-koodi, salvestada automaatselt GPS-koordinaadid ja lisada metaandmeid (sügavus, kirjeldus). Andmed sünkroonitakse turvalise JWT-autentimisega REST-API kaudu ühisesse PostgreSQL-andmebaasi, kust need on kohe kättesaadavad ka veebirakendusele.

Süsteemi töökindlust kinnitavad ulatuslikud üksus- ja integratsioonitestid. Projekti tulemusena on täppisväetamise-aparatuuri toetav tarkvaraplatvorm turvalisem, paremini hooldatav ja märksa kasutajasõbralikum, aidates seeläbi kaasa mulla tasakaalustatud ja jätkusuutlikule väetamisele.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 35 leheküljel, 6 peatükki, 14 joonist, 3 tabelit.

Abstract

Precision Fertilization Web and Mobile Application

This bachelor's thesis focuses on modernizing a six-year-old precision fertilization web application — originally scaffolded with JHipster — and on creating a companion mobile app that streamlines soil-sample collection and analysis in the field.

At the heart of the project is a capillary electrophoresis device that determines soil composition so farmers can fertilize precisely and avoid over-fertilization. While the device performs chemical analysis, users must supply contextual data such as sampling coordinates. The legacy desktop tool for this task has been retired; all interaction now happens through the web or the new mobile application, which also offers access to historical results and statistics.

To extend the life of the web platform, the existing Angular/Spring Boot codebase was audited and upgraded to the latest framework versions, eliminating security risks and version conflicts. The architecture was refactored into cleanly separated front-end and back-end modules for easier maintenance.

The React Native mobile app lets users scan a sample bag's QR code, automatically capture GPS coordinates, and attach metadata (depth, notes, photos). Data sync securely over a JWT-protected REST API to a central PostgreSQL database, making new records instantly available in the web interface.

Comprehensive unit and user tests attest to the system's robustness. The resulting software platform is more secure, maintainable, and user-friendly, supporting balanced and sustainable soil fertilization.

The thesis is in Estonian and contains 35 pages of text, 6 chapters, 14 figures, 3 tables.

Lühendite ja mõistete sõnastik

API	Rakendusliides (*Application Programming Interface*)
CI/CD	Pidevintegratsioon/Pidevvalmidus (*Continuous Integration/Continuous Delivery*)
CRUD	Andmebaasi põhioperatsioonid (*Create, Read, Update, Delete*)
GET-päring	HTTP-meetod, mis küsib andmeid serverilt
GPS	Ülemaailmne positsioneerimissüsteem (*Global Positioning System*)
HTTP	Hüperteksti edastusprotokoll (*HyperText Transfer Protocol*)
HTTPS	Turvaline hüperteksti edastusprotokoll (*HyperText Transfer Protocol Secure*)
ID	Identifikaator
IP-aadress	Internetiaadress (*Internet Protocol address*)
JWT	Internetistandard andmete loomiseks koos digitaalallkirjaga (*JSON Web Token*)
QR-kood	Kahemõõtmeline maatrikskood info talletamiseks
REST	Esitusoleku siire (*Representational State Transfer*)
SSL	Turvasoklikiht (*Secure Sockets Layer*)
TLS	Transpordikihi turbeprotokoll (*Transport Layer Security*)
USB	Universaalne jadasiin (*Universal Serial Bus*)

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Põldude üleväetamine on jätkuvalt suureks probleemiks üle kogu maailma. ÜRO toidu- ja põllumajandusorganisatsiooni sõnul ei jõua keskmiselt iga teine kilo väetist taimedeni [1]. Suur osa ülejäägist ei jää mulda, vaid leostub vette, lendub ammoniaagina või eraldub kasvuhoonegaaside kujul. Sellel on omakorda tagajärjed. Üleväetamine toob endaga kaasa põhjavee ja veekogude reostumise. Ligi 70–90% lämmastiku ja 60–80% fosfori hajureostusest Läänemeres on põhjustatud põllumajanduse poolt [2]. Lisaks keskkonnajalajäljele on üleväetamine majanduslik kulu, mida saaks vältida.

Üleväetamist on võimalik vältida, väetades põllumaad vastavalt taimestiku vajadusele. Vajaliku väetisekoguse kindlakstegemine vajab pinnaseanalüüse. Mulla analüüsimise teenust pakuvad erinevad laborid. Selleks tuleb võtta iseseisvalt mullaproov ning saata see postiga laborisse. Analüüside hinnaklassiks on paarkümmend kuni sada paarkümmend eurot proovi kohta. Proovide analüüsimine teostatakse vähem kui nädalaga, millele lisandub proovide transpordiks kulunud aeg. Põllumaa analüüsimiseks vajalik proovide hulk võib samuti varieeruda paarist kuni paarikümneni. See võib omakorda tähendada mitme tuhande eurost väljaminekut [3–5].

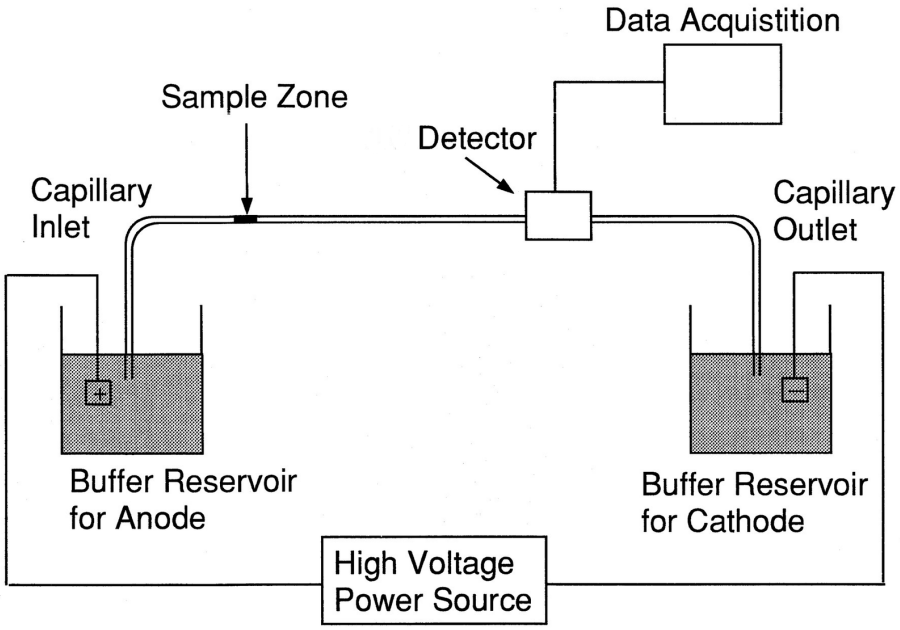
Täppisväetamise protsessi hõlbustamiseks on Tallinna Tehnikaülikooli teadlased Keemia ja biotehnoloogia instituudist loonud tehnoloogilise lahenduse. Lahenduse keskseks osaks on kapillaarelektroforeesi teostav kaasaskantav masin (vt Joonis 1). Masina abil on võimalik iseseisvalt analüüsida mullaproove. Analüüsitud proovide tulemused saadetakse kesksesse serverisse, kus määratakse mullas leiduvad analüüdid ning nende kogused. Analüüsitud mulla koostisega saab tutvuda veebirakenduse abil.

Joonis 1. SMAGRY-nutrients masin[1].

1.1 Kuidas toimib kapillaarelektroforeesi seade?

Kapillaarelektroforees on analüütilises keemias kasutatav meetod, mille abil on võimalik elektrivälja toimel eraldada üksteisest erinevaid ioonseid analüüte [6]. Masinas on kaks puhverlahusega täidetud anumat, mis on ühendatud õhukese klaas- või kvartskapillaariga. Lahustele rakendatakse kõrgepinget, mille toimel hakkavad analüüdid kapillaaris liikuma. Analüüdid liiguvad vastavalt oma laengule ja suurusele eri kiirusel mööda kapillaari detektorini, kus registreeritakse analüütide kontsentratsioonile vastavad signaalid ehk piigid. Piikide järgi on võimalik kindlaks määrata proovis leiduvad ained ja nende kogused. Masina struktuuri illustreerib Joonis 2.

[1] Pilt võetud lehelt `https://smagry.com/`



Joonis 2. Kapillaarelektroforeesi tööpõhimõte[2].

1.2 Veebi- ja mobiilirakendused

Varasemalt on üliõpilased arendanud antud seadmele töölaua- ja veebirakenduse. Arendus on toimunud nelja lõputöö ning mitme ülikooli aineprojekti raames [7–10]. Aja jooksul loobuti töölauarakendusest ning kasutati vaid veebirakendust. Paraku olid veebirakenduse raamistikud aegunud ning veebirakendus polnud avalikult kättesaadav turvalisuse kaalutlustel.

Tellija soovib veebirakendusele lisaks mobiilirakenduse loomist. Mobiilirakenduse eesmärgiks on lihtsustada mullaproovide võtmisel lähteandmete salvestamist, mis aitab proovide analüüsimisel mullaproove identifitseerida. Mobiilirakendus pakub alternatiivi veebirakendusele analüüsitulemuste vaatamiseks.

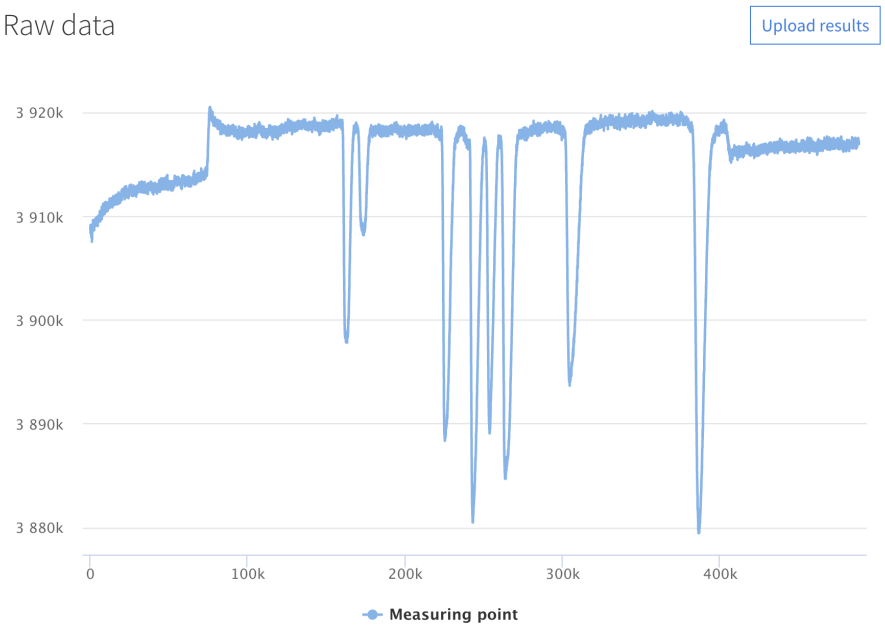
[2]Pilt võetud lehelt `https://en.wikipedia.org/wiki/File:Capillary_Gel_Electrophoresis_Instrument_Schematic.png`

2 Eesmärk

Lõputöö eesmärk on kaasajastada olemasolevat tarkvara, et seda oleks võimalik kasutajatele taas kättesaadavaks teha. Kuna aastatega on toimunud olulisi muutusi seoses seadme ja töölauarakendusega, tuleb ka süsteemi arhitektuuri osaliselt ümber kujundada sujuva töö tagamiseks.

2.1 Varasema mulla analüüsimisprotsessi kirjeldus

Pinnase analüüsimiseks tuleb võtta mullaproov ning jätta meelde proovi asukoht. Analüüsi tegemiseks ja tulemuste salvestamiseks oli kasutuses eelnevalt mainitud töölauarakendus. Tulemused kujutasid endast töötlemata andmeid, mida saab esitada graafikuna. Graafikul saab näha piike (vt Joonis 3), mille abil on võimalik hiljem määrata mulla koostis. Tulemuste salvestamisel saadetakse need ka veebirakendusse, kus on võimalik teha lõplik analüüs ja määrata mullas leiduvad ained. Veebirakenduse kaudu on lisaks võimalik vaadata ja hallata kõiki varasemaid analüüse ja nende tulemusi.

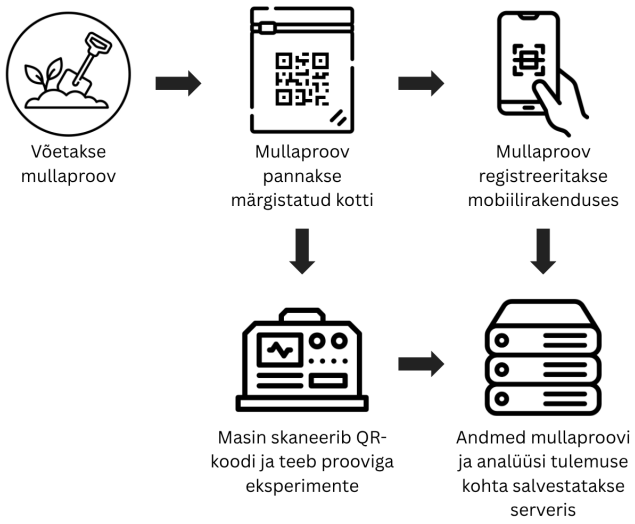


Joonis 3. Masina saadetud toorandmete graafik veebirakenduses.

2.2 Uue analüüsimisprotsessi kirjeldus

Aastatel, mil veebirakendust ei hooldatud, arendati kapillaarelektroforeesi masinat edasi. Uusim masina versioon ei vaja töötamiseks töölauarakendust, kuna tal on olemas oma sisseehitatud tarkvara ning suhtlemine veebirakendusega käib otse läbi veebirakenduse API lõpp-punktide. Töölauarakenduse arendamisest ja kasutamisest on täielikult loobutud. Uuendatud masinale on lisatud ka QR-koodide skanner, eesmärgiga pakkuda uut funktsionaalsust mullaproovide haldamise ja analüüside tegemise lihtsustamiseks.

Protsessi lihtsustamisel mängib kõige olulisemat rolli kavandatav mobiilirakendus. Proovi võtmisel pannakse see QR-koodiga märgistatud koti sisse. Mobiilirakendus võimaldab kasutajal vahetult peale proovi võtmist skaneerida QR-koodi, sisestada vajalikke andmeid proovi kohta ning saata need API-le. Kasutaja näeb kõiki oma võetud proove mobiilirakenduses. Kui masin hiljem analüüsib vastavat mullaproovi, saadab ta API-le analüüsi tulemused koos loetud QR-koodiga, võimaldades API-l viia tulemused kokku õige mullaprooviga. Kui analüüs on läbi viidud, saab kasutaja näha tulemusi nii mobiili- kui ka veebirakenduses. Rakenduse tööprotsess on kujutatud Joonisel 4.



Joonis 4. Analüüsiprotsessi diagramm.

2.3 Varasem veebirakendus

Esialgne kapillaarelektroforeesi veebirakendus oli loodud kuus aastat tagasi bakalaureusetöö raames ning seda täiendati aasta hiljem järgneva lõputöö käigus. Hiljem on rakendust

arendatud mitmete aineprojektide jooksul erinevate üliõpilaste poolt. Veebirakendus on loodud JHipsteri koodigeneraatoriga [11]. Rakendusse oli loodud iga järgneva arendustsükli käigus uusi funktsionaalsusi, kuid polnud pööratud tähelepanu raamistike ja teekide kaasajastamisele, mille versioonid pärinesid rakenduse esmasest loomisest 2019. aastal. Veebirakendus polnud ülikooli võrgu väliselt kättesaadav ning kasutas HTTP-protokolli.

2.3.1 Veebirakenduse puudused

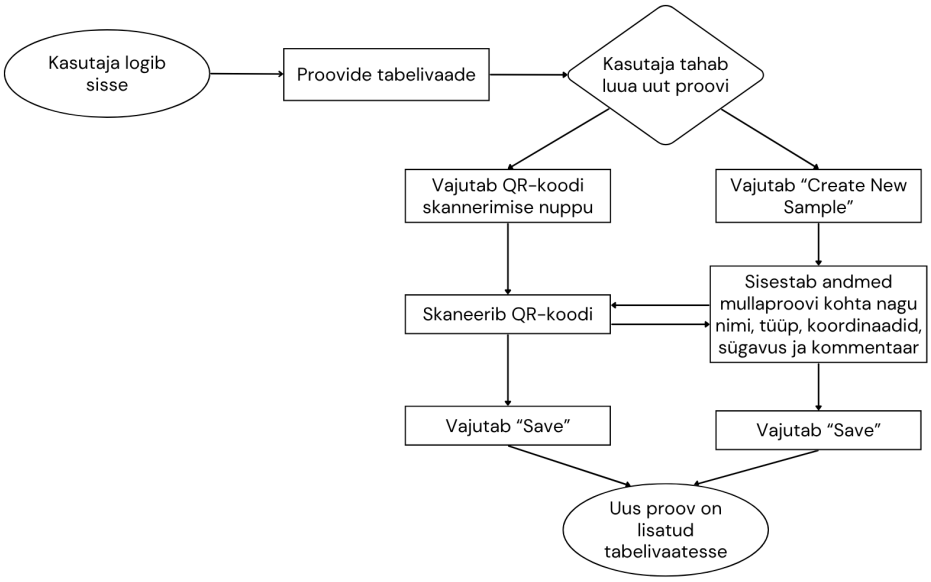
Veebirakenduse hoolduse hooletusse jätmine nende aastate jooksul on toonud kaasa mitmesuguseid probleeme:

- **Turvalisuse oht:** vananenud teekide kasutamisega kaasnevad turvariskid, kuna neil on teadaolevaid haavatavusi, mida pahatahtlikud osapooled võivad proovida ära kasutada [12].
- **Jõudlusprobleemid:** võrreldes uuemate tehnoloogiatega võivad vananenud teegid olla vähem tõhusad. Lisaks tehakse suur osa andmetöötlusest eesrakenduses.
- **Kokkusobimatus tänapäevaste tööriistadega:** vananenud teegid ei pruugi uuematega töötada, mis takistab mugavat arendusprotsessi.
- **Tehniline võlg:** kogunenud vananenud teegid tekitavad hapraid süsteeme, mida on riskantne muuta, mille tõttu on tulevasi uuendusi luua keerulisem ja aeganõudvam [13].
- **Kasutajamugavus:** eesrakendus kasutab JHipsteri koodigeneraatori standardset kujundust, mis on kasutajale kohmakas.
- **Aeglus:** eesrakendus kasutab API-ga suheldes liigselt mahukaid päringuid, mis aeglustavad rakenduse tööd (seda käsitletakse hiljem detailsemalt).

2.4 Mobiilirakendus

Varasem lahendus proovide võtmiseks, analüüsimiseks ja salvestamiseks oli kasutajale ebamugav. Sageli tuleb mulla analüüsimisel võtta mitmeid proove, mida on hiljem keeruline üksteisest eristada. Mullaproovide koheseks registreerimiseks oli välitöödele vaja võtta arvuti, mis osutus ebamugavaks. Proovide hilisemal töölauarakendusega registreerimisel võisid proovid omavahel vahetusse minna ning seetõttu saadi hiljem vale ülevaade põldude väetamisvajadusest. Tellija soovis, et mullaproovide registreerimine ja andmete sisestamine

toimuks läbi spetsiaalse mobiilirakenduse, sest mobiiltelefoni on mugavam välitöödel kasutada. Mullaproove saab registreerida spetsiaalsete proovikottide QR-koode telefoni kaameraga lugedes. Proovikoha koordinaadid määratakse, kasutades nutitelefoni sisseehitatud GPS-i. Vajadusel saab kasutaja lisada ka täiendavaid andmeid. Seejärel saadetakse andmed edasi veebiserverisse. Lisaks toodi välja, et antud mobiilirakenduses võiks interaktiivse kaardi peal näha olla kõikide varasemate proovide asukohad ning võimalus hallata kasutaja varasemaid proove ja analüüside tulemusi. Näite eduka proovi loomise voost saab näha Joonisel 5.



Joonis 5. Proovi loomise vooskeem.

Kokkuvõttes tuli luua mobiilirakendus, kuhu salvestatakse mullaproove, kasutades nende koti QR-koode. Proovide analüüsimisel skaneeritakse mullaproovid masina QR-koodi skanneriga. Analüüsi valmides saadetakse tulemused koos QR-koodiga veebiserverisse, kus need viiakse kokku eelnevalt salvestatud proovi parameetritega.

2.5 Töö eesmärgid

Tellijaga arutades olid toodud välja esialgsed eesmärgid antud lõputöö jaoks:

- viia veebirakendus üle uusimale JHipsteri versioonile, et saaks kasutada teekide uuemaid versioone;

- luua veebirakendusele lisaks mobiilirakendus, mille eesmärgiks on lihtsustada mullaproovide halduse ja registreerimisprotsessi;
- olemasoleva veebirakenduse kasutatavuse parandamine;
- veebirakenduse olemasolevate funktsionaalsuste säilitamine;
- interaktiivse kaardi lisamine olemasoleva andmestiku põhjal, et kasutajal tekiks ülevaade oma võetud mullaproovidest.

3 Analüüs

Antud peatükk esitab põhjalikuma analüüsi varasemate rakenduste struktuuridest, funktsionaalsustest ning kasutatud tehnoloogiatest. Kirjeldatakse planeeritud arendusi ning põhjendatakse arendusprotsessi käigus tehtud otsuseid.

3.1 Varasem tarkvaraline lahendus

Varasemalt seadmega töötamiseks pidi kasutaja kapillaarelektroforeesi masina ühendama arvutiga USB-kaabli abil. Seadme juhtimiseks oli loodud töölauarakendus, kus kasutaja sai sisestada vajalikud katseparameetrid. Katse käivitamisel võis tulemusi reaalajas jälgida ning need salvestati lokaalselt. Võrguühenduse olemasolul edastati katsetulemused veebirakendusele. Kogu süsteemi arhitektuur on kujutatud Joonisel 6.

Töölauarakenduse ülesanded:

- **Füüsiline side:** seadme Arduino-põhine juhtplokk on USB-kaabli kaudu otse ühendatud töölauarakendusega, kust andmed loetakse reaalajas.
- **Juhtnupud rakenduses:** “Start”, “Stop”, “Salvesta”, “Puhasta” ning eraldi kõrgepinge lüliti.
- **Automaatne taimer:** määratud aja täitumisel lõpetab töölauarakendus katse, salvestab failid ja lülitab kõrgepinge välja.
- **Andmete edastamine:** pärast katset talletatakse mõõtmised kohalikku arvutisse ja edastatakse need veebirakendusele edasise analüüsi ja aruandluse tarbeks.

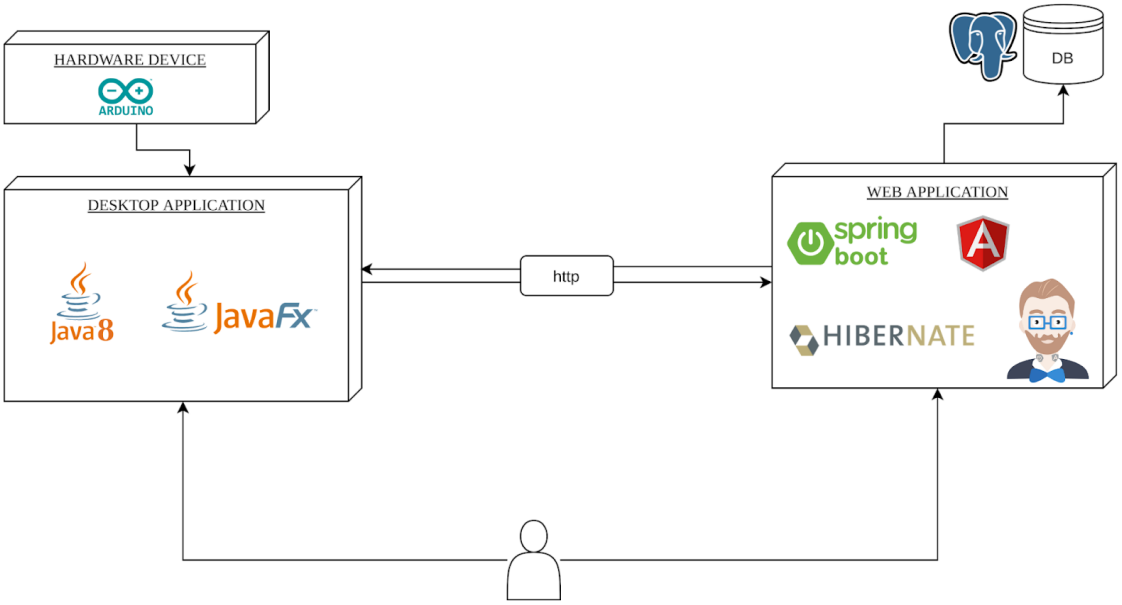
Aastate jooksul on täppisväetamise masinat edasi arendatud ning kogu töölauarakenduses olnud funktsionaalsus on masinasse sisseehitatud. Tänu sellele on loobutud töölauarakenduse kasutamisest ning arendamisest.

Veebirakendus oli varem monoliitne: ees- ja tagarakendus paiknesid samas koodibaasis ning töötasid ühe protsessina TalTechi serveris. Samas serveris jooksis ka PostgreSQLi

andmebaas, kuhu salvestati töölauarakendusest pärinevad mõõtetulemused. Rakenduse põhiülesanne oli analüüsimasina katsetulemuste põhjal tuvastada analüüdid ja määrata nende kontsentratsioonid.

Veebirakenduse ülesanded:

- **Katsete vastuvõtt ja talletamine:** võtab töölauarakendusest tuleva eksperimendi-JSONi vastu REST-API kaudu ja salvestab kõik toor- ning metaandmed PostgreSQLi [14] andmebaasi.
- **Automaatne signaalitöötlus:** puhastab mõõtesignaali, leiab piigid ja arvutab nende pindalad, et tavakasutaja saaks kohe näha esialgseid tulemusi.
- **Analüütide tuvastus ja käsitsi kinnitamine:** sobitab leitud piigid ainete nimekirjaga ning lubab teadlasel need vajadusel käsitsi üle määrata ja versioonihalduses talletada.
- **Kalibratsioonikõvera ja detektsioonipiiride arvutamine:** loob lineaarse regressiooni vähemalt kolmest positiivsest katsest, arvestab ühikute teisendusi ning annab usalduspiirid ja avastamispiirid.
- **Riskipiiride hindamine:** võrdleb mõõdetud kontsentratsioone ainete ohupiiridega.
- **Raportite genereerimine:** koostab PDF-raporteid kahes kujus: detailne teadlasele (metoodika, detektsioonipiirid, regressioonid) ja lihtne tavakasutajale (kontsentratsioonid, ohutasemed).
- **Kasutajaliides ja andmevaated:** pakub Angular-põhiseid [15] tabeleid filtrite, lehekülgimise ja Highcharts-graafikutega [16], sh eksperimendi- ja piiginimekirjad ning metoodikate haldus.
- **Analüüsi- ja meetodiparameetrite haldus:** salvestab kasutaja (või meetodi) vaikimisi parameetrid, võimaldades neid hiljem uuesti rakendada ja võrrelda.
- **Rollipõhine ligipääs:** eristab teadlase ja tavakasutaja õigusi (JWT-põhine autentimine [17]): teadlane loob meetodeid ja kinnitab analüüse, tavakasutaja viib läbi standardseid katseid.



Joonis 6. Varasem projekti disain[1].

3.1.1 Kasutatud tehnoloogiad

Veebirakendus loodi algselt JHipsteri abil, mis genereeris Spring Bootil [18] põhineva tagarakenduse ning Angulariil põhineva eesrakenduse. Andmebaasina kasutati PostgreSQLi ning andmebaasimudeli versioonihalduseks Liquibase'i [19].

JHipsteri eelis seisneb võimes kiiresti luua täisfunktsionaalseid veebirakendusi, kasutades kaasaegseid tehnoloogiaid ja arenduse parimaid tavasid. Siiski, pikaajalise arenduse jooksul selgus, et automaatselt genereeritud koodiga kaasnevad mitmed piirangud, mis teevad raskemaks süsteemi edasist arendamist ja hooldamist.

JHipster koodigeneraator

JHipster võimaldab genereerida valmisprojekte, mis ühendavad eesrakenduse, tagarakenduse ja andmebaasi, kas monoliitseks rakenduseks või eraldiseisvateks rakendusteks. Selline lähenemine sobib hästi kiireks prototüüpimiseks, kuid toob kaasa arhitektuurilisi jäikusi:

- **Struktuurne jäikus:** genereeritud koodi muutmine võib osutuda keeruliseks, kuna sõltuvuste ja struktuuride omavaheline seotus ei pruugi olla piisavalt modulaarne.
- **Ajale jalgu jäämine:** koodigeneraatorid ei pruugi kiiresti kohaneda tehnoloogiate

[1]Pilt võetud lõputööst "Kapillaarelektroforeesi andmetöötluse töölaua- ja veebirakendus."

arenguga, mistõttu võib genereeritud kood kiiresti vananeda.

- **Loovuse piiramine:** koodigeneraatorite kasutamine võib takistada arendajate võimalust rakendada oma kogemustest ja projekti vajadustest lähtuvaid lahendusi.

JHipsteril on oma domeenikeel, mille abil saab kasutaja defineerida olemeid, nende vahelisi suhteid, valideerimisreegleid jne. JHipsteri koodigeneraatorisse on sisseehitatud funktsionaalsus genereeritud koodi uuendamiseks. Paraku töötab antud funktsionaalsus vaid juhul, kui äriloogika on domeenikeeles kirjeldatud. Koodigeneraator ei suuda eristada genereeritud ja arendaja poolt hilisemalt lisatud koodi ning kirjutab arendaja poolt tehtud muudatused üle.

3.2 Vajalikud edasiarendused

Esmane prioriteet veebirakenduse edasiarendusel oli uuendamine. Aegunud tehnoloogiad võivad kujutada endast reaalset turvaohtu, mistõttu ei saanud veebirakendust produktsiooni viia ilma nendega tegelemata. Samuti oli vaja rakenduse API-osasse viia sisse teatud muudatused, et mobiilirakenduse arendust alustada.

3.2.1 Uuendamise strateegia ja struktuuri ümberkujundamine

Esialgne valitud strateegia oli uuendada veebirakendust kasutades selleks JHipsteri koodigeneraatori käske `jhipster upgrade` ja `jhipster-migrate` [20]. Need on spetsiaalsed käsud mõeldud uuendama JHipsteriga loodud veebirakendusi ühelt JHipsteri versioonilt teisele, uuendades korraga ka rakenduses kasutatavaid muid sõltuvusi. Veebirakenduse uuendamine selle strateegiaga sai alguse 2024. aasta suvel, kui üks antud lõputöö autoritest alustas suvepraktika raames veebirakenduse arendamist. Uuendamist jätkati antud lõputöö autorite koosseisus õppeaines Tarkvaraarenduse tellimusprojekt.

2024. sügise algul oli uusim JHipsteri versioon 8.7.1, rakendus ise toetus versioonile 7.0.0. Sellest versioonist edasisaamisega tekkis oluliselt probleeme. Oli mitmeid katseid edasi uuendada veebirakendust, aga iga katsega ilmnes suur hulk koodikonflikte ja muid tõrkeid, mille lahendamine nõudis rohkem aega, kui oleks mõistlik. Projektihalduse lihtsustamiseks oli vaja uuendamise strateegiat muuta.

Ühe võimalusena kaaluti arendust täiesti puhtalt lehelt. See pakuks suurt paindlikkust nii

tehnoloogiate valikul kui ka rakenduse arhitektuuri kujundamisel. Sellise tee valimisel tulnuks aga kogu äriloogika uuesti üles ehitada või taastada – ülesanne, mis eeldab projekti sügavamat tundmist ning muudaks töö üleliia keerukaks ja ajamahukaks.

Strateegia, millega projekt lõpuks edasi liikus, kujunes välja JHipsteri uuendamisel tekkinud probleemide lähemal uurimisel. JHipsteri uuendamiskäsud tegelesid paralleelselt kõikide sõltuvustega, mis veebirakenduses kasutatud (v.a. need, mida arendajad olid hiljem ise lisanud). Kui uuendamisprotsessi käigus tekivad koodis konfliktid või muud vead, siis arendaja saab nendest teadlikuks vaid juhul, kui protsess on lõppenud ning vead kuhjunud. Antud olukorra teeb eriti probleemseks veebirakenduse monoliitne struktuur, mis ühendab endas nii taga- kui ka eesrakendust. Eesrakenduse mittetöötamise korral vea tõttu ei käivitu tagarakendus, ja vastupidi.

Võttes arvesse nimetatud kitsaskohti, otsustati üle minna modulaarsele arhitektuurile, kus serveripoolne tagarakendus ja kliendipoolne eesrakendus töötavad eraldiseisvate teenustena. See teeb projektihalduse selgemaks ja arendustsükli kiiremaks. Selleks jagati senine JHipsteri-põhine monoliit kaheks rakenduseks ja loobuti raamistikust kui tarbetust sõltuvusest, säilitades samas olemasoleva äriloogika. Tehnoloogiauuendusi saab nüüd teha järk-järgult, lahendades versioonikonfliktid kohe nende ilmnemisel. Arhitektuuri suurim eelis peitub rikete piiritletuses – ühe komponendi tõrge ei mõjuta teise tööd. See on eriti oluline mobiilirakenduse arendamisel, sest eraldiseisev API jääb kättesaadavaks ka siis, kui veebikliendi kasutajaliides ajutiselt rivist väljas on.

Pärast algversiooni valmimist lisati veebirakendusse mitu uut funktsionaalsust, kuid JHipsteri koodigeneraator ei osanud neid uuenduste käigus säilitada ja kustutas need mõnikord sootuks. Raamistikust loobumine kõrvaldas selle riski: kõik lisafunktsioonid püsivad nüüd alles ning arendajad saavad neid vajadusel turvaliselt uuendada ja edasi arendada.

Kokkuvõttes pakkus veebirakenduse kaheks eraldamine kaks tähtsamat eelist:

- **Lihtsam hooldus:** komponentide eraldatus võimaldab neid sõltumatult uuendada ja asendada.
- **Selgem vastutus:** iga komponent täidab kindlat funktsiooni ja suhtleb teistega läbi määratletud liideste (nt REST API).

3.2.2 API ja andmebaasi täiendused

Veebirakenduse API on kapillaarelektroforeesi tarkvaraarhitektuuris võtmetähtsusega, sest see on vajalik nii eesrakenduse kui ka mobiilirakenduse tööks. API vahendab suhtlust andmebaasiga: kasutajaliidesed saadavad API-le päringuid ning API tagastab andmebaasist andmeid. Seega sõltub kasutajaliideste töö suures osas API-st ja andmebaasist.

Täiendused mobiilirakenduse jaoks

Mobiilirakenduse peamine funktsioon on mullaproovide haldus. Selleks peab mobiilirakendus andmebaasist kätte saama mullaproovide olemeid, mis kuuluvad kasutajale, ja nendega seotud eksperimentide ja analüüside tulemusi. Projekti alguses ei olnud see võimalik, sest mullaproovid kui olemitüüp ei eksisteerinud ei andmebaasis ega API-s. Olemitüübid olid olemas eksperimentide, analüüside jm nendega seotud info jaoks.

Selleks, et mobiilirakendus saaks oma eesmärki täita, oli vaja andmebaasi lisada tabel aineproovide jaoks ning tekitada seos eksperimentide ja proovide vahel, et kasutajal oleks võimalik proovidele tehtud eksperimentide tulemusi näha. API-sse oli vaja ka vastavaid lõpp-punkte lisada, kuhu mobiilirakendus saaks päringuid saata.

Täiendused eesrakenduse jaoks

Kuna eesrakendus oli arendatud koos tagarakendusega, siis API-l oli põhimõtteliselt kõik eesrakenduse töö jaoks vajalik juba olemas, kuid nende omavahelisel suhtlusel oli siiski probleeme. Eesrakenduse uuendamise järel pandi tähele, et kindlatel lehtedel on rakenduse töö oluliselt aeglasem. Võrguliikluse uurimisel selgus, et veebirakendust aeglustasid ebavajalikult mahukad API-päringud. Filtrid, mille konstrueerimiseks oli enamasti vaja vaid paar infovälja, kasutasid kohandamata GET-päringuid, mis tagastasid kordades rohkem andmeid, kui praktiliselt vaja oli.

Filtrid vajavad enamasti ainult objekti ID-d, mida kasutada API-le uue päringu tegemiseks, ja objekti nime, mida filtrimenüüs kuvada, et kasutaja saaks aru, millega tegemist. Mõni filtri rippmenüü kuvab vaid objektide ID-sid (nt eksperiment-filter analüüside lehel) – sellisel juhul on vaja pärida vaid unikaalsete ID-de loendit. API-sse oli vaja selliseid päringuid juurde kirjutada, et eesrakendus saaks kiiremini töötada.

3.3 Mobiilirakenduse kavandamine

Mobiilirakenduse arendus vajas antud projektis eriti põhjalikku etteplaneerimist kahel tähtsamal põhjusel:

- Erinevalt veebirakendusest ei olnud mobiilirakendust varasemas arhitektuuris olemas. Tegemist oli tarkvaraga, mida tuli täielikult nullist arendada, mis tähendas, et rakenduse struktuuri, äriloogikat, välimust ja kasutatavaid tehnoloogiad pidi põhjalikult läbi mõtlema ja kavandama enne, kui võis alustada koodi kirjutamist.
- Ühelgi lõputöö autoril polnud enne antud projekti kogemust mobiilirakenduse arenduses. Seetõttu nõudis planeerimine ka erilist ettevaatlikkust, et vältida kogenematusest tulenevaid vigu, ning lisaks tuli aega pühendada veel uute tehnoloogiate õppimiseks.

Planeerimisetapis käis tihe suhtlus tellijaga, et kontrollida mobiilirakenduse kavandi vastavust nõuetele. Esmalt tehti otsus, millist tarkvararaamistikku kasutada rakenduse loomiseks. Seejärel disainiti mobiilirakendusele prototüüp, millest lähtuti hiljem koodi kirjutamisel.

3.3.1 Nõuded mobiilirakendusele

Kõik tellija poolt täpsustatud nõuded mobiilirakendusele tuginesid peamiselt ühele mobiilirakenduse tähtsaimale funktsionaalsusele – mullaproovide haldamisele. Nimetatud funktsionaalsus määrab mobiilirakenduse rolli analüüsimisprotsessis. Mobiilirakendus peab pakkuma kasutajale mugavat viisi identifitseerida võetud proove, hallata nende andmeid ning jälgida nende analüüsitulemusi.

Mobiilirakenduse nõuded olid järgmised:

1. Mobiilirakendus pidi tuginema samale API-le ja andmebaasile, mida kasutab veebirakendus, jagades sellega nii kasutajakontosid kui ka mullaproovide andmeid.
2. Kasutajal pidi olema võimalik rakenduse kaudu uusi proove salvestada ning kõiki oma loodud proove vaadata, muuta ning kustutada.
3. Mobiilirakendus pidi võimaldama QR-koodide skaneerimist. Igale mullaproovile vastab unikaalne QR-kood, mille järgi saab seda proovi identifitseerida. Igal

salvestatud proovil peab olema seotud QR-kood.

4. Kasutajal pidi olema võimalus oma loodud proovide seast kindlaid proove otsida.

3.3.2 Tehnoloogia valik

Mobiilirakenduse arenduseks kaaluti kahte raamistikku: React Native [21] ja Flutter [22]. Need on kaks populaarsematest mobiilirakenduste arenduse raamistikest. Mõlemad võimaldavad ristplatvormilist arendust ehk lubavad luua rakendusi, mis töötavad nii iOS kui ka Android süsteemidel. Nimetatud raamistike populaarse staatuse tõttu on nende kohta olemas palju õppematerjale ja laialdane kogukonna tugi. Kahest valikust osutus sobivaimaks React Native.

Flutteril eeliseks on, et sellega loodud rakendused on valdavalt parema jõudlusega kui React Native'iga loodud rakendused, kuid React Native'i omadused olid antud projekti jaoks sobivamad. Esiteks on React Native varem loodud tehnoloogia, mistõttu on sellel rohkem kogukonna tuge ning materjale. Teiseks oli React Native'i raamistik loodud autoritele tuttava JavaScripti keele põhjal, mistõttu pakkus see mõnel määral tuttavamat loogikat. Flutter kasutas seevastu Darti programmeerimiskeelt, millega tutvumiseks oleks kulunud rohkem aega.

Oli tähtis, et arenduseks kasutatavate tehnoloogiate omandamiseks ei kuluks liigselt aega, et projekti eesmärgid saaksid õigeks ajaks täidetud. Seega React Native'i võrdlemisi algajale sõbralikumad omadused ja keskkond olid otsustavad faktorid selle kasuks.

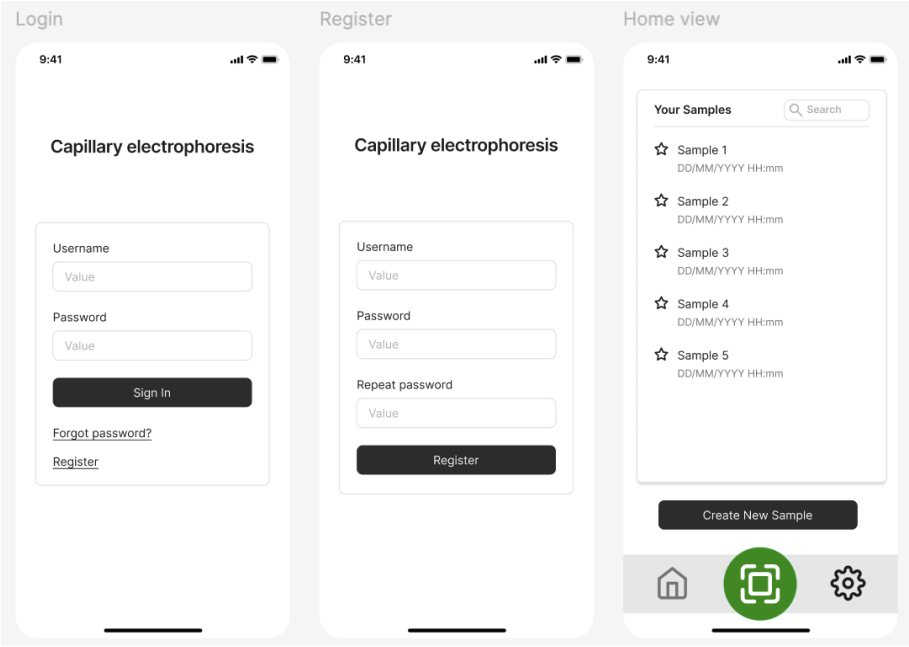
3.3.3 Prototüübi disain

Mobiilirakenduse jaoks loodi prototüüp Figma [23] keskkonnas (vt Jooniseid 7, 8). Disaini loomisel arvestati kasutusmugavuse põhimõtteid nagu selgus ja lihtsus. Disainietapis oli fookus sellel, et proovide loomine, vaatamine, muutmine ning kustutamine oleksid võimalikult mugavad ja lihtsad.

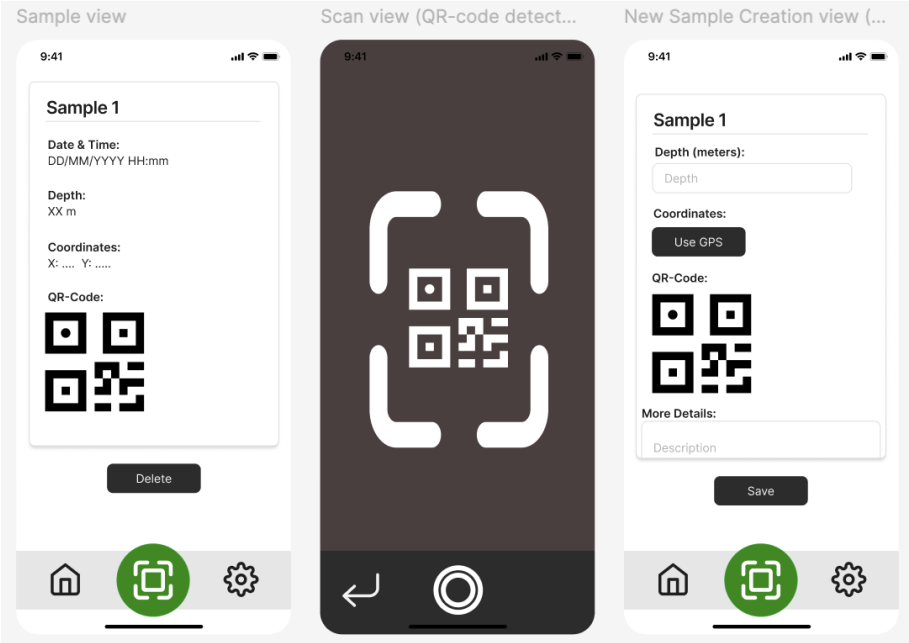
Kui kasutaja logib sisse, näeb ta esimese asjana tabelit oma võetud proovidest. Proove saab sorteerida mitmel viisil ning otsida ka nime järgi. Uue proovi skaneerimiseks peab vajutama alumisel navigatsiooniribal olevat rohelist nuppu, mis avab telefoni kaamera. Kui kaamera tuvastab QR-koodi, avab rakendus vaate proovi andmete sisestamiseks. Kui kasutaja on

andmed salvestanud, saadetakse need serverisse ning uus proov ilmub kasutajale proovide tabelisse. Navigatsiooniriba võimaldab kasutajal lihtsasti liikuda proovide tabeli, QR-koodi skanneri ja rakenduse sättete vahel.

Rakenduse kujundus oli tehtud minimalistlikult. Domineeriv värv on valge, tekst ning nupud on kas mustad või hallid. Kui mingi element mobiilirakenduses nõudis rohkem tähelepanu või eristumist, kasutati selle jaoks rohelist, mis tundus olevat loogiline valik seoses mulla väetamise temaatikaga. Näiteks ütleb erkroheline katseklaasi ikoon proovi kõrval, et seda on laboris analüüsitud, ning navigatsiooniribal on suur roheline QR-koodi skannerimise nupp.



Joonis 7. Figma kavandid sisselogimis-, registreerimis- ja tabelivaatest.



Joonis 8. Figma kavandid proovi detailvaatest, skannerist ja proovi loomisvaatest.

3.4 Arendusprotsess

Arendustöö järgnes Agile-metoodikale [24], mis võimaldas paindlikku ja järk-järgulist süsteemi loomist vastavalt juhendaja tagasisidele. Agile lähenemine sobib hästi tarkvaraarenduse projektidele, kus täpsed nõuded kujunevad välja töö käigus ning on oluline kiire kohanemisvõime muutuvate oludega.

3.4.1 Iteratiivne arendus

Projekti jooksul viidi arendust läbi nädalapikkuste sprintidena, mille käigus seati igale tiimiliikmele konkreetsed ülesanded (nt uus funktsionaalsus, vigade parandamine vms). Iga iteratsiooni lõpus tehti koosolek juhendajaga, et hinnata tehtud tööd ning leppida kokku järgmised sammud.

3.4.2 Tööde haldamine GitLabis

Töö organiseerimiseks kasutati Taltechi GitLabi. Oli otsustatud luua eraldi repositooriumid eesrakendusele, tagarakendusele, mobiilirakendusele ja pöördproksile. Loodud repositooriumid koondati olemasolevasse projekti gruppi, kus paiknesid antud projektiga seotud varasemalt loodud rakendused. GitLabis loodi iga ülesande jaoks eraldi *issue* ehk pilet, millele lisati kirjeldus ja vastutaja. Iga piletiga seotud arendus viidi läbi eraldi harus enne

põhiharusse lisamist.

Kasutatud töövoog järgis loogikat:

- Iga töö algas pileti loomisest (näiteks uue funktsiooni kirjeldus või vea raport).
- Loodi vastav haru, mille nimetus vastas pileti numbrile ja sisule.
- Peale töö lõpetamist esitati liitmistaotlus (*merge request*), mille käigus vaadati kood üle ja testiti muudatusi.
- Kui muudatus vastas nõuetele, ühendati see põhiharusse.

Mahukamate tööülesannete täitmisel, näiteks raamistike kaasajastamisel, koondati töövoog ühe pileti alla. Raamistike uuendamisel oleks iga alamülesande jaoks eraldi pileti loomine muutnud versioonihalduse oluliselt kohmakamaks. Selle asemel kirjeldati alamülesanded ning tehnilised lahendused versioonisammude (*commit*'ide) kirjeldustes ning liitmistaotluste kommentaaridena.

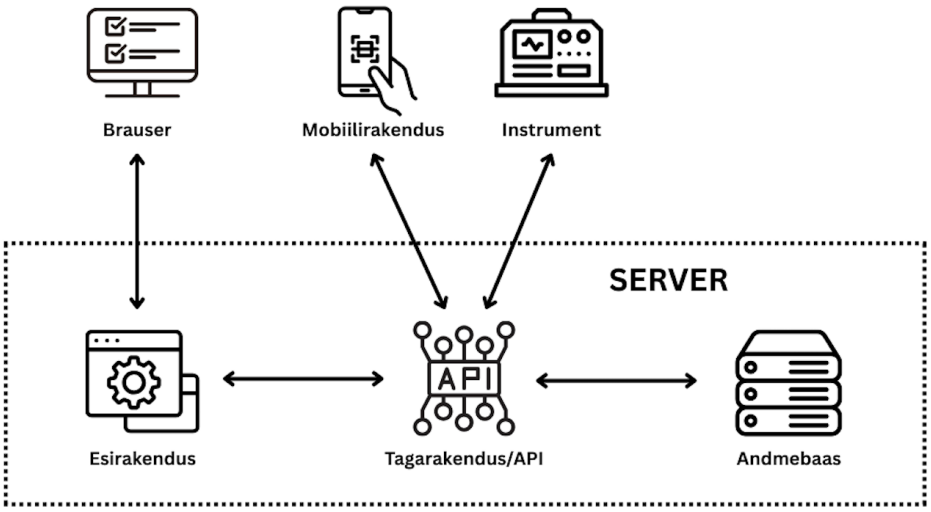
4 Tulemused

Peatükk kirjeldab rakenduse moderniseerimise teekonda – kuidas varasem JHipsteri-põhine monoliit jagati eraldiseisvaks tagarakenduseks, eesrakenduseks ja andmekihiks. Lühidalt avatakse arhitektuurimuudatuse ajend, tehnoloogilised valikud ning suurimad võidud, milleks on töökindlam API, hõlpsam versioonihaldus ja alus mobiilirakenduse loomiseks.

4.1 Arhitektuur

Rakenduse uuendamisel oli üheks eesmärgiks modulaarne disain (vt Joonis 9). Oli otsustatud luua iga veebirakenduse osa jaoks eraldiseisev Dockeri [25] konteiner. See lähenemine võimaldas selgelt piiritleda iga teenuse vastutusalad ning vähendas ristmõjust tulenevaid riske.

Pöördproksi, eesrakendus, tagarakendus ja andmebaas said koondatud eraldiseisvatesse konteineritesse. Iga konteineris jooksev teenus sai oma versioonihalduse, mis tähendab, et iga rakenduse osa saab uuendada ilma teisi osi mõjutamata. Lahendus on eriti mugav veebirakenduse ja mobiilirakenduse paralleelsel arendamisel, sest mõlemad kasutavad sama API-t ja andmebaasi.



Joonis 9. Rakenduste suhtediagramm.

4.2 Andmebaas

Lõputöö käigus oli tehtud olemasolevast PostgreSQL andmebaasist eraldi koopia, mida uuendatud rakendused kasutavad. Nii jääb algne andmebaas puutumatuks ja vajadusel saab lähteseisu taastada. Kõik skeemimuudatused rakendati migratsioonitööriista Liquibase'i abil.

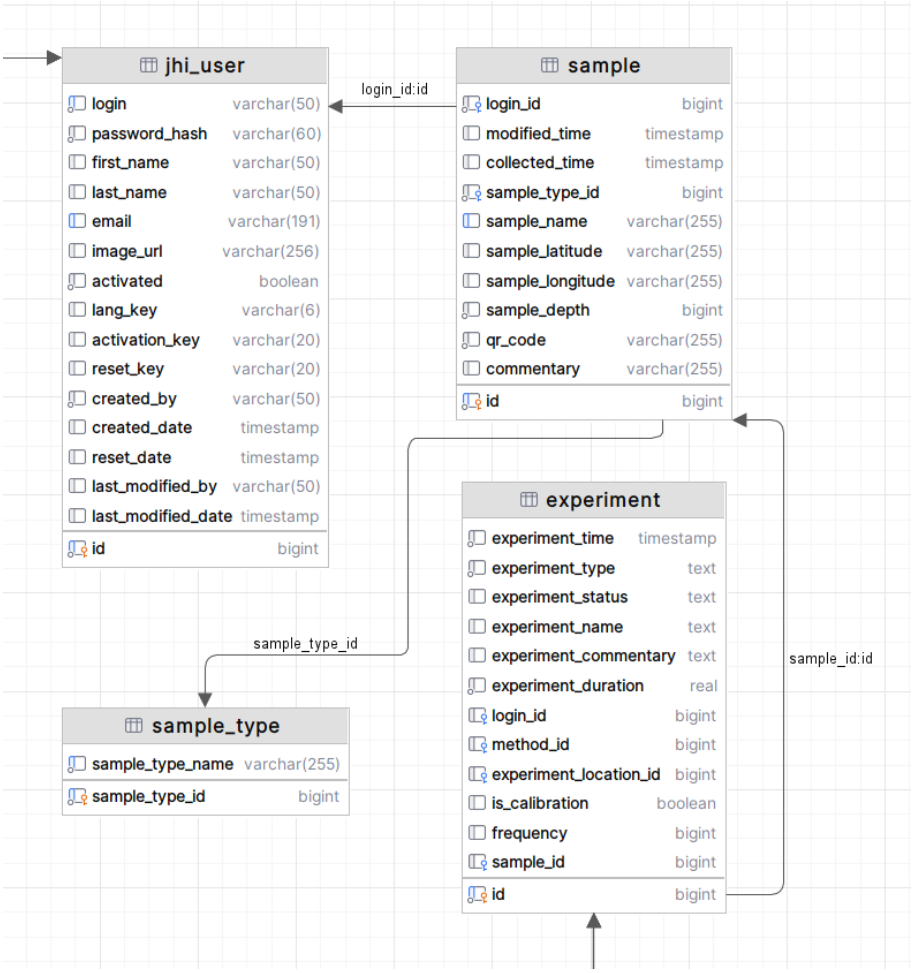
4.2.1 Aineproovide tabel

Aineproovide (*Sample*'ite) tabel puudus algse andmebaasi tabelite seast, seega oli vaja see juurde luua. Lisaks tuli luua klassifikaatorite tabel, et määrata proovidele tüübid (vt Joonis 10).

Kohustuslikeks väljadeks proovi juures olid määratud:

- login ID;
- proovi tüüpi ID;
- võtmise sügavus;
- QR-kood.

Lisaks on aineproovil olemas loomise ja muutmise ajatemplid, nimi, laius- ja pikkuskraad ning kommentaar.



Joonis 10. Proovi ja proovi tüübi tabelid ning nendega seotud tabelid.

4.2.2 Aineproovide suhe eksperimentidega

Aineproovide ja eksperimentide vahel on üks-mitmele seos (ühel proovil mitu eksperimenti, aga eksperimendil ainult üks proov). Andmebaasis lisati proovi primaarvõti eksperimendi tabelisse välisvõtmeks. Spring Booti *Sample* olemitüübi külge lisati eksperimentide loend, mille kaudu seotakse andmebaasist kõik vastava proovi eksperimendid.

4.3 Rakenduste eraldamine

Ees- ja tagarakenduse eraldamiseks tuli mõlema rakenduse failid tõsta eraldi repositooriumitesse. Varasema monoliitse rakenduse käivitamine käis läbi Maveni [26] `pom.xml` konfiguratsioonifaili. Rakenduse eraldamisel tuli antud failist eemaldada kõik eesrakendusega seonduv.

4.4 Tagarakendus

Tagarakenduse API eraldamine ja püstisaamine oli esmane prioriteet, et nii eesrakendus kui ka mobiilirakendus saaksid andmeid kätte. Seetõttu oli vaja API võimalikult kiiresti töötavale kujule saada ning serverisse püsti panna. See lihtsustas eesrakenduse ja mobiilirakenduse arendamist, sest enam ei pidanud arendusprotsessi käigus käivitama oma arvutis tagarakendust.

4.4.1 Rakenduste uuendamine

Spring Booti raamistiku uuendamine versioonilt 2 versioonile 3 tõi kaasa hulga ebakõlasid teiste teekidega. Versioonimuutuse käigus liikus nimeruum Javaxilt Jakartasse, mis põhjustas mitmete teekidega konflikte. Muutusid ka mitmed Spring Booti autentimise ja päringute haldamise funktsionaalsused. Seetõttu pidime Spring Security konfiguratsiooni ümber kirjutama.

Tagarakenduses JHipsterist loobumisel puudus lihtne lahendus. Otsustasime vajalikud JHipsteri failid teegist oma projekti tõsta. JHipsteri lähtekoodi kasutamine on lubatud, kuid projekti avaldamine nõuab Apache 2.0 litsentsi kasutamist. Antud litsents lubab vaba kasutust (sh ärilistel eesmärkidel), koodi muutmist ja sulgemist. Lisasime litsentsi oma projekti juurkausta.

4.5 Pöördproksi

Turvalise ja sujuva infovahetuse tagamiseks sai kogu liikluse ette pandud NGINX-il [27] põhinev pöördproksi (vt Lisa 2). Kõik HTTP-päringud (port 80) on suunatud kohe 301-ga HTTPS-ile, et välistada krüpteerimata ühendused. Proksi lõpetab TLS-i, haldab SSL-sertifikaate ja lubab ainult TLS 1.2–1.3 koos tugevate šifritega. Tänu sellele saab sisemine võrk suhelda lihtsas HTTP-s, samal ajal kui privaatvõtmed püsivad eraldatud turvatsoonis.

Liiklus jaguneb kahel moel: juure URL-ile (/) tulevad päringud jõuavad eesrakenduseni, samas kui kõik */api*, */services*, */management* ja teised sarnased teed suunatakse tagarakendusse klastrisse. Proksi lisab päringutele ka *X-Real-IP* ja *X-Forwarded-For* päised, et API saaks kliendi IP-aadressi logida, piirata või autentida. Kui tulevikus peaks juurde tekkima uusi API-sid, saab need lihtsalt lisada *upstream*-plokki ja NGINX jagab koormuse

automaatselt, võimaldades teenuseid katkestusteta uuendada ja skaleerida.

4.6 Eesrakendus

Kui tagarakendus ja NGINX-i pöördproksi olid serverisse paigaldatud, sai alustada eesrakenduse arendust. Arendaja võis küll kasutada serveris töötavat API-d, kuid enamasti tuli ees- ja tagarakendust korraga muuta ja testida. Sellisel juhul käivitati mõlemad rakendused lokaalselt. Serveri poolel suunab HTTP-päringud NGINX, ent lokaalarenduseks oli lisatud eesrakendusse eraldi proksikonfiguratsioon, mis aktiveerub automaatselt siis, kui veebirakendus käivitatakse arendusrežiimis.

4.6.1 Uuendamine

Eesrakendus tuli uuendada Angulari versioonilt 11 versioonile 18, mis oli lõputöö algushetkel värskeim stabiilne väljalase. Algul prooviti uuendada üle mitme versiooni korraga, kuid üheaegselt tekkivate konfliktide hulk muutis töö ebapraktiliselt keeruliseks. Lõpuks sai valitud iteratiivne lähenemine: liiguti ühe suure versioonisammu kaupa, käivitades iga etapi järel käsu `ng update`, mis uuendas peamised sõltuvused ja refaktoreeris osa koodi automaatselt. Seejärel paigaldati paketid uuesti ja ülejäänud konfliktid lahendati käsitsi.

Kõrvalekallete lahendamine oli sageli ajamahukas, sest teekide dokumentatsioon ei näidanud alati selgelt, milline versioon konkreetse Angulari väljalaskega ühilduks. Mitmel kolmanda osapoole teegil kadus uuemate Angulari versioonide tugi sootuks, mistõttu asendati need alternatiividega või Angulari ametlike moodulitega. Varasematesse versioonidesse lisatud abiteekide hulk oli peamine põhjus, miks konflikte palju tekkis.

Peale iga konfliktide vooru testiti rakendust põhjalikult ning töö jätkus samas rütmis järgmise versioonini, kuni lõpuks jõuti Angular 18-ni.

Kõik versioonimuutused on esitatud tabelis 1. Suurimad muudatused on järgmised:

- Dragula tugi lõppes. Meetodi-protseduuri lohistus funkstionaalsus kirjutati ümber kasutades Angular CDK Drag & Drop teeki;
- Angulari deklareerimata (*standalone*) komponentide kasutuselevõtt;
- Ametliku HighChartsi pakendi kasutamine ja graafikute migreerimine;
- TypeScript 5 ja range režiim sundisid lisama täpsemaid TypeScripti tüüpe, mis

parandas koodi kvaliteeti.

Tabel 1. Eesrakenduse tähtsaimate versioonimuutuste võrdlus.

Pakett / väli	**Vana versioon**	**Uus versioon**
@angular (core jms)	11.2.x	18.2.13
@angular/cli	11.2.4	18.2.18
TypeScript	4.1.5	5.4.5
Bootstrap / Bootswatch	4.6.0	5.3.2
@ng-bootstrap/ng-bootstrap	9.0.2	17.0.1
Webpack	4.46.0	5.76.1
ESLint	7.22.0	9.27.0
Prettier	2.2.1	3.3.3
Jest	26.6.3	29.7.0
Node (engines)	$\geq$ 14.16.0	$\geq$ 18.20.0

4.6.2 Vead kasutajaliideses ja nende parandamine

Muudatused Angulari versioonide vahel põhjustasid mitmeid kujunduse vigu eesrakenduses. Kõige märgatavamad olid rakenduses kasutusel olevate stiiliraamistike Bootstrapi [28] ja Bootswatchi [29] versiooni vahetusest tingitud stiilinihked, mis segasid veebilehe töövoogu. Kasutajaliides oli üles ehitatud, kasutades Bootstrapi versiooni 4, aga see viidi rakenduses üle versioonile 5 Angulari versiooniga kooskõlastamiseks. Versioonide üleminekul olid paljud eesrakenduses kasutatud elemendid Bootstrapist kaotatud, nii et oli vaja uurida dokumentatsiooni, et leida sobivaid asendusi lehe välimuse taastamiseks.

Oli ka palju vigu, mis pärinesid veebirakenduse vanast versioonist. Kõige probleemsem neist oli seoses kustutamisnuppudega. Kustutamisnupud olid olemas peaaegu igas tabelivaates. Iga tabeli olemi juures kõige parempoolsemas tulbas on tavaliselt kolm nuppu: *View, Edit, Delete* (Vaata lähemalt, Muuda, Kustuta). Nupud võivad erineda sõltuvalt olemitüübist, aga enamikul on see kolmik. Kustutamisnupp on ainus nupp, mis avab hüpikakna, ning on seetõttu teistest nuppudest teistmoodi struktureeritud.

Probleem seisnes selles, et kui klikkida kustutamisnupule, andis veebileht veakoodi 404, mis näitas, et tegu oli marsruutimisveaga. Viga oli olemas igal kustutamisnupul

sõltumata vaatest. Koodi uurides selgus, et kuigi kustutamisnuppude marsruudid olid `*.route.ts` failides defineeritud, ei olnud need Angular Routerile (Angulari rakenduste navigatsioonihaldur) edasi antud, mistõttu ei osanud rakendus seostada kustutamisnuppude poolt antud marsruute hüpikakendega. Selle parandamisega oli võimalik veebirakenduses olemeid taas kustutada.

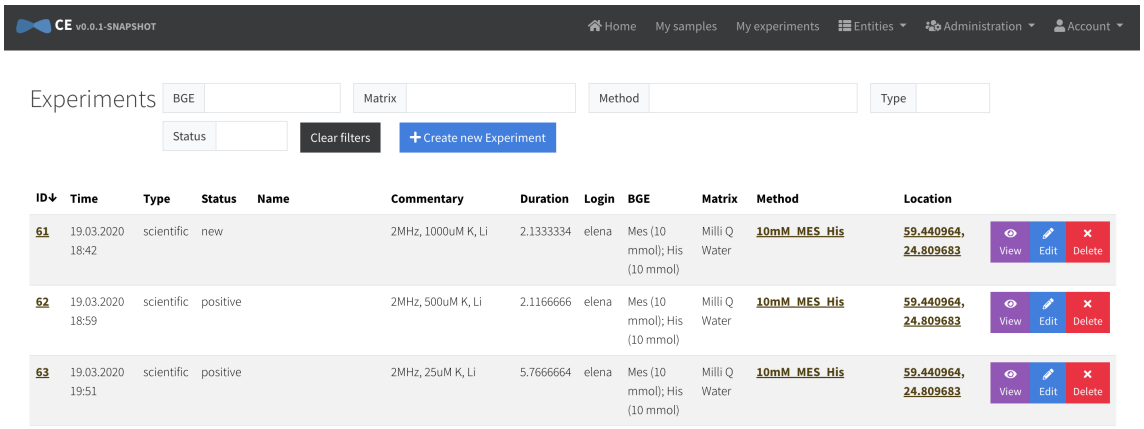
4.6.3 Optimeerimine

Eesrakenduse tööd aeglustasid peamiselt mahukad `GET`-päringud API-le, millest suurem osa tehti tabelivaadete filtrite konstrueerimiseks. Tabelid ja filtrid (vt Joonis 11) on antud veebirakenduses väga levinud, nii et kasutajal on raske eirata nende poolt tekitatud aeglustust.

Kõige mahukamad filtrite päringud olid *Method, Analysis, Experiment* ja *Analyte* andmetüüpide puhul, mis võisid pärida kümnetes kordades rohkem andmeid, kui neil vaja oli. *Analysis* filter võttis näiteks keskmiselt 3 sekundit aega ning laadis alla 3,2 MB andmeid, millest kasulik info filtri jaoks moodustas vaid 0,21%.

Antud veebirakenduse filtrite konstrueerimiseks on veebilehel vaja teada vastava andmetüübi kõiki unikaalseid väärtusi andmebaasis. Valdavalt ei ole vaja kogu infot iga väärtuse kohta. Vajalik on info, mida filtris kuvada, et kasutaja saaks väärtuseid eristada (tavaliselt nimi), ning indikaator, mille järgi andmete filtreerimine toimub (tavaliselt ID). GET-päringud, mida veebirakendus kasutas filtrite jaoks, pärisid aga kõiki andmeid tabelist, sest spetsiifilisemaid päringuid polnud API-s defineeritud.

Tagarakendusse oli seega vaja juurde lisada kitsamad GET-päringud iga eelnimetatud andmeklassi jaoks. Oli tähtis, et need päringud koguksid vaid vajalikest väljadest andmeid (ID ja nime väljad), et andmebaas liigselt aega otsimisele ei kulutaks. Eesrakenduse kood tuli ümber kirjutada, et filtrid teeksid õigeid päringuid ning jälgida muutusi. Tänu sellele säilitasid filtrid muudetud päringutega oma korrektse funktsionaalsuse ja töötasid märgatavalt kiiremini.



Joonis 11. Näide tabelivaatest veebirakenduses.

4.6.4 Aineproovide lisamine

Kasutaja peab saama hallata oma proove ka veebirakenduses. Seetõttu loodi:

- **My Samples** vaade tavakasutajale (analoogne **My Experiments** lehele);
- Administraatorivaade, mis kuvab kõik proovid andmebaasist;
- Detail-, muutmis- ja kustutamisvaated iga proovi jaoks.

4.7 CI/CD

Muudatuste automaatseks laadimiseks ja rakendamiseks kasutatakse Dockeri põhist GitLab CI/CD Runnerit. Eesrakenduse, tagarakenduse ja pöördproksi jaoks on seadistatud eraldi andmekonveierid, mis käivituvad alati, kui arendaja liidab oma haru *main*-haruga. Serveris toimub seejärel koodi kompileerimine ja konteinerite käivitamine. Eesrakenduse ja tagarakenduse puhul eelneb *deploy*le ka automaatne testimise etapp.

4.8 HTTPS ja SSL-sertifikaat

Tellija nõudis, et veebirakendus oleks ülikooli välisest võrgust vabalt kättesaadav ning kliendi-serveri suhtlus toimuks turvaliselt HTTPS-protokolli kaudu. Selleks taotleti ülikoolilt veebirakenduse domeenile SSL-sertifikaati, mis paikneb serveri failisüsteemis. See on projekti ainus komponent, mis ei tööta Docker-konteineris. Pöördproksi (NGINX) pääseb sertifikaadile ligi ja lõpetab TLS-ühenduse serveri piiril.

4.9 Mobiilirakendus

Mobiilirakenduse arendus toimus paralleelselt veebirakenduse arendusega. Arenduses kasutati Expo ja Expo Go [30] tööriistu, mis pakkusid mugavat keskkonda rakenduse kompileerimiseks ja testimiseks.

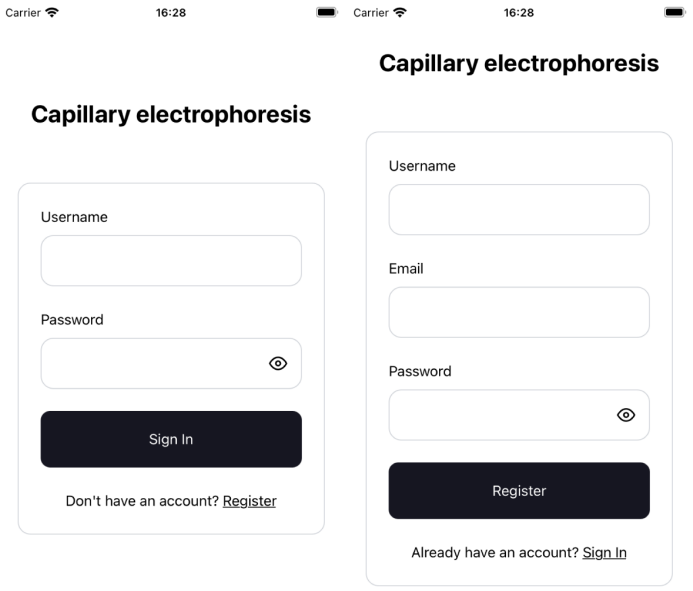
Kasutajaid on võimalik autentida ning registreerida mobiilirakenduses kasutades veebirakenduse API-t. Eduka sisselogimise järel salvestab mobiilirakendus enda mällu kasutaja identifikaatori ning API-lt saadud JWT pääsme, mida ta kasutab edaspidistes API-päringutes.

4.9.1 Mobiilirakenduse vaated

Mobiilirakenduse vaated olid implementeeritud enamjaolt nii nagu prototüübis ettenähtud. Esines vaid väikseid visuaalseid erinevusi. Hiljem lisandus arendusprotsessis nõue mobiilirakenduses eksperimentide ja analüüside tulemuste kuvamiseks. Selle jaoks implementeeriti eraldi vaade võttes aluseks veebirakenduses eksperimendi detailvaated.

Mobiilirakenduse vaadete loetelu:

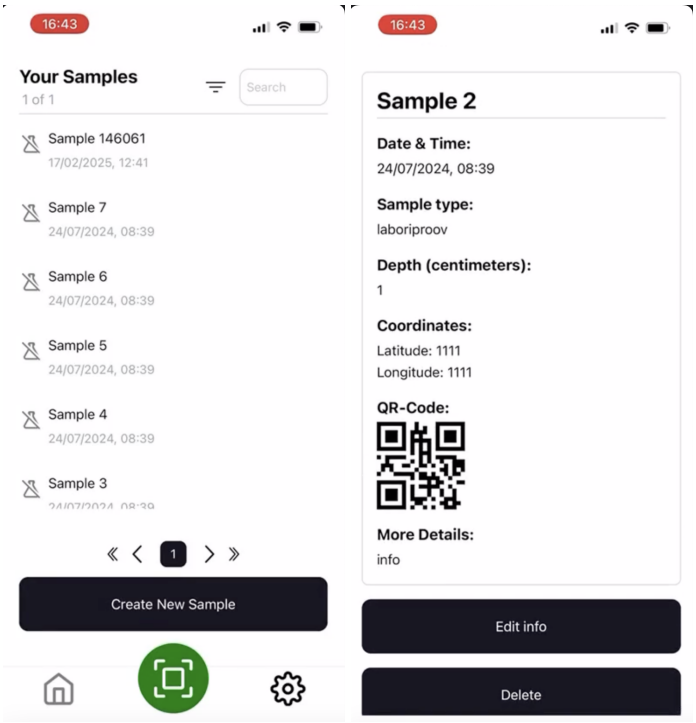
- **Sisselogimise vaade:** kasutajanime väli, salasõna väli, link uue kasutaja registreerimiseks (vt Joonis 12a).
- **Registreerimise vaade:** meiliaadressi väli, kasutajanime väli, salasõna väli, link sisselogimiseks (vt Joonis 12b).



(a) (b)

Joonis 12. Mobiilirakenduse sisselogimis- ja registreerimisvaated.

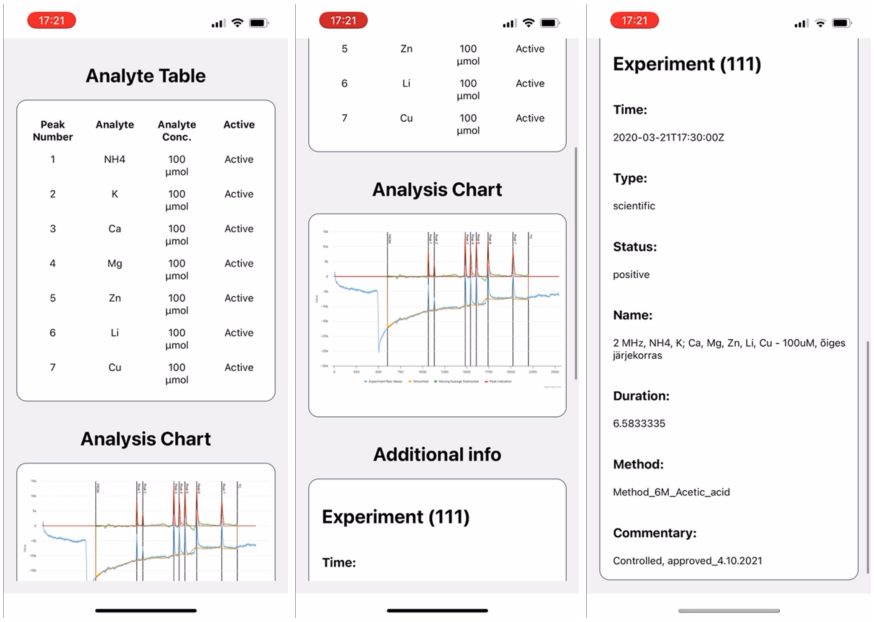
- **Aineproovide vaade:** tabel aineproovidest, iga aineproovi juures nimi, registreerimise aeg ja kõrvalolev katseklaasi ikoon, mis olenevalt värvist näitab, kas proovi on analüüsitud. Tabeli kohal on nupp tabeli sorteerimiseks ja otsinguväli, tabeli all on nupp uue aineproovi salvestamiseks (vt Joonis 13a).
- **Aineproovi detailvaade:** näitab aineproovi kogu infot ning laseb kasutajal proovi infot muuta. Kui prooviga pole veel katseid tehtud on kasutajal võimalik proovi veel kustutada. Kui prooviga on katseid tehtud, siis ei saa seda enam kustutada, kuid on võimalik vaadata katsete tulemusi (vt Joonis 13b).



(a) (b)

Joonis 13. Mobiilirakenduse proovide tabel ning näide proovi detailvaatest.

- **Eksperimendi detailvaade:** kui eksperimendil on analüüsi andmed, siis ilmuvad detailvaatesse esimesena analüütide graafik (Joonisel 14a) ja tabel (Joonisel 14b), eksperimendi muu info on alati kuvatud (Joonisel 14c).



(a) (b) (c)

Joonis 14. Mobiilirakenduse eksperimendi detailvaade.

4.9.2 Aineproovide sidumine eksperimentidega

Mobiilirakenduses:

- Proovil, millel on eksperimente, kuvatakse loendis vasakul roheline katseklaasi ikoon; ilma eksperimentideta proovil hall ikoon.
- Eksperimendiga proovi ei saa kustutada; detailvaates asendab kustutamisnuppu eksperimentide loend.
- Kui eksperimendil on analüüsi tulemused, kuvatakse piikide graafik (HighCharts + React-Native-Web-View) ning analüütide tabel; muu info on alati nähtav.

4.10 Refleksioon

Tähtsamad lõputöö käigus püstitatud eesmärgid said täidetud. Nii veebirakendus kui ka mobiilirakendus jõudsid valmis kujule ning veebirakendus on internetist ligipääsetav. Aja puudujäämise tõttu jäi tegemata vaid interaktiivse kaardi lisafunktsionaalsus, mille puudumine ei pärsi veebirakenduse tööd. Mobiilirakenduse arendamine kulges ilma suuremate probleemideta. Kõige enam tekkis tõrkeid seoses vana veebirakenduse uuendamise ja täiendamisega.

4.10.1 Veebirakenduse uuendamine

Uuendusprotsessi käigus selgus, et kuigi oli arvestatud veebirakenduse sõltuvuste ajamahuka uuendamisega, sai tegelikku töömahtu alahinnatud. Eriti projekti algusfaasis osutus vajalikuks peale iga suuremat sammu uuesti hinnata tööde ulatust ja kohandada edasisi tegevusplaane, võttes arvesse uue rakenduse arhitektuuri ja vajadusi.

Positiivse aspektina tuleb esile tuua, et hoolimata märkimisväärsest vajadusest koodi refaktoreerimiseks, säilis vana rakenduse äriloogika. Eriti oluline oli, et eksperimentidest saadud andmete analüüsi algoritm jäi uuenduste käigus muutumatuks, mis tagas rakenduse põhifunktsionaalsuse järjepidevuse.

Ühe peamise õppetunnina tuleb esile asjaolu, et märkimisväärset ajakulu oleks olnud võimalik vältida, kui oleks varakult otsustatud loobuda JHipsteri kasutamisest. Esialgne eeldus, et JHipsteri koodigeneraator suudab hõlbustada versiooniuuendusi, osutus praktikas

ebatäpseks ning loodetud tööriist osutus vähem tõhusaks kui algselt eeldatud.

4.10.2 Soovitused vanade rakenduste uuendamisel

Vanade rakenduste uuendamisel on oluline pöörata tähelepanu järgmistele aspektidele:

- **Rakenduse struktuuri ja äriloogika mõistmine:** Enne uuenduste alustamist tuleks põhjalikult tutvuda rakenduse sisemise ülesehituse ja äriloogikaga. Eriti oluline on mõista API päringute toimimist, lõpp-punktide tähendust ning objektide struktuuri. Mida paremini on arendaja rakenduse toimimisest teadlik, seda lihtsam on läbi viia edasisi muudatusi ja siluda võimalikke probleeme.
- **Iteratiivne sõltuvuste uuendamine:** Sõltuvusi ja teeke on soovitatav uuendada järk-järgult, näiteks üksikult või väikeste gruppidena. Peale iga muutust on mõistlik rakendust testida, et võimalikult varakult tuvastada tekkinud vead. Selline lähenemine võimaldab lahendada konfliktid kohe nende ilmnemisel ning vähendab võimalike probleemide kuhjumist tulevikus, parandades kogu süsteemi hooldatavust.
- **Sõltuvuste kokkusobivuse kontroll:** Praktikas võib sageli juhtuda, et mõned vanad teegid ei ühildu enam uute versioonidega või teiste sõltuvustega samas projektis. Sellisel juhul tuleb otsida sobivaid alternatiive ja vajadusel refaktoreerida olemasolev kood. Seetõttu on oluline juba varakult teha teadlikke valikuid kasutatavate sõltuvuste osas, kuna need mõjutavad otseselt tulevasi hoolduskulusid ja süsteemi tehnilise võla ulatust.

4.10.3 Mobiilirakenduse arendus

Mobiilirakenduse loomine kulges valdavalt ilma probleemideta. Suurim takistus oli kogenematus mobiilirakenduste arendamises, mis aeglustas kohati tööprotsessi. React Native'i valik oli mõistlik, sest kohanemiseks ei kulunud liigselt aega. Mobiilirakendus jõudis valmis kujule projekti lõpuks.

5 Valideerimine

Veebirakendust ja mobiilirakendust valideeriti peamiselt automatiseeritud testidega. Mobiilirakenduse puhul saadi samuti tellija poolt tagasisidet.

5.1 Veebirakenduse töö valideerimine

Veebirakenduse valideerimiseks kasutati peamiselt üksus- ning integratsiooniteste. Tagarakenduses olid üksustestid kirjutatud JUnit 5 raamistikuga [31] ning integratsioonitestid Spring Test Context raamistikuga [32]. Eesrakendus kasutas testide jaoks Jest teeki.

Veebirakenduse testid olid varasemast ajast juba olemas, kuid need olid samuti aegunud, mistõttu oli neid vaja vastavalt projektis toimunud uuendustele muuta ning kirjutada teste juurde uute funktsionaalsuste jaoks.

Tagarakenduses on 346 testi, mis kontrollivad kõiki üksuseid, autentimist, kasutajate haldamist jm REST API funktsionaalsust. Testide koodikatvus REST API puhul on 83%. Eesrakenduses on 357 testi, mis kontrollivad kasutajaliidese komponentide struktureerimist ja käitumist ning suhtlust API-ga. Nende koodikatvus on 68%.

5.2 Eesrakenduse töö kiirendamine

All on võrreldud algsete ja muudetud päringute aja- ja andmemahtu. Taga- ja eesrakendused pandi tööle lokaalses masinas. Katsete vahel ei toimunud andmebaasis muutuseid. Andmete vaatlus oli tehtud, kasutades Chrome DevTools tööriista [33]. Võrreldi muutuseid päringute lõpule viimiseks kulunud ajas ning allalaetud andmete mahtu. Kuna ühe ja sama päringu aeg võib varieeruda olenevalt võrgu kiirusest, vaadeldi iga päringu puhul mõõtmiste keskmist. Andmemahu puhul ei olnud varieerumist, kuna päritavad andmed olid alati samad.

Tabelis 2 on välja toodud keskmised ajad filtrite algsetel ning uuendatud päringutel ja nende vahe näitamaks, kui palju aega on päringu kohta säästetud. Tabel 3 näitab, mitu

kilobaiti andmeid tõmbavad uued päringud võrreldes algsete päringutega ning kui suure osa moodustab uus maht vanast.

Tabel 2. Algsete ja muudetud päringute ajaline võrdlus.

Klass	**Aeg enne (ms)**	**Aeg pärast (ms)**	**Vahe (ms)**
Method	271	106	165
Analysis	3080	38	3042
Experiment	547	119	428
Analyte	164	83	81

Tabel 3. Algsete ja muudetud päringute mahu võrdlus.

Klass	**Andmemaht enne (kB)**	**Andmemaht pärast (kB)**	**% algsest mahust**
Method	142	4	2,82
Analysis	3241	6,9	0,21
Experiment	4443	198	4,46
Analyte	54,6	9,2	16,85

5.3 Mobiilirakenduse töö valideerimine

Mobiilirakenduse valideerimiseks kirjutati Jest [34] üksusteste ning koguti kasutajapoolset tagasisidet. Teste on 57 ning need katavad 82% koodist. Testitakse rakenduse komponentide korrektset struktureerimist. Arendusprotsessi käigus käis pidev suhtlus tellijaga, et saada tagasisidet tehtud töö kohta ning kasutajakogemust paremaks teha. Valminud rakendus on saanud tellija poolt heakskiidu.

6 Kokkuvõte

Antud lõputöö eesmärgiks oli kaasajastada olemasolevat täppisväetamise veebirakendust ning luua sellele juurde mobiilirakendus, mis võimaldaks efektiivsemat andmete sisestust ja vaatamist liikuva töö kontekstis. Töö käigus jagati monoliitne veebirakendus eraldiseisvateks ees- ja tagarakendusteks. Monoliitne JHipsteri-põhine arendusraamistik eemaldati ning Angular uuendati versioonile 18, mis parandas süsteemi jõudlust, hooldatavust ja kasutajakogemust. Samuti taastati eesrakenduse algne kujundus, säilitades kasutajatele tuttava töövoo. Veebirakenduse aadress on olemas Lisas 3.

Lõputöö käigus loodi mobiilirakendus mullaproovide haldamiseks. Kasutajal on võimalik lisaks uute mullaproovide sisestamisele ka vaadata seotud analüüside tulemusi otse mobiilirakendusest. Rakendus on loodud arvestades praktilist kasutust põllumajandus- ja laboritöös.

Süsteemi töökindluse tagamiseks taastati ja täiustati olemasolevad testid nii ees- kui ka tagarakenduses ning uute funktsionaalsuste jaoks kirjutati täiendavad testid. Mobiilirakenduse jaoks loodi kõik testid nullist, et tagada selle usaldusväärne töö.

Kokkuvõttes sai enamik eesmärkidest täidetud, veebirakendus ja mobiilirakendus jõudsid mõlemad valmis kujule ning loodud lahendus parandab oluliselt täppisväetamise töövoogu. Edasiseks arenduseks saaks implementeerida planeeritud lisafunktsionaalsused, millega antud lõputöö käigus ei jõutud tegeleda – mullaproovide kaardivaade. Samuti võib veebirakendust veelgi kiirendada ning parandada selle turvalisust.

Kasutatud kirjandus

[1] ÜRO toidu ja põllumajandusorganisatsioon. *New data to measure cropland nutrient budgets*. [Kasutatud 29.04.2025]. URL: https://www.fao.org/newsroom/detail/new-data-to-measure-cropland-nutrient-budgets/en.

[2] Läänemere merekeskkonna kaitse komisjon. *Helsinki: Baltic Marine Environment*. [Kasutatud 02.05.2025]. URL: https://helcom.fi/action-areas/agriculture/.

[3] Eesti Keskkonnauuringute Keskus. *Pinnase ja tahke aine analüüside hinnad alates*. [Kasutatud 02.05.2025]. URL: https://www.klab.ee/wp-content/uploads/2024/01/pinnaseanalyyside_hinnad_2024.pdf.

[4] Maaelu Teadmuskeskus. *Agrokeemia valdkonna teenuste hinnakirjad*. [Kasutatud 02.05.2025]. URL: https://metk.agri.ee/hinnakirjad#agrokeemiateenused.

[5] Linas Agro. *Yara Megalab™ mullaanalüüsid*. [Kasutatud 02.05.2025]. URL: https://linasagro.ee/agro-uudised/yara-megalab-2024.

[6] LibreTexts Chemistry. *Capillary Electrophoresis*. [Kasutatud 29.04.2025]. URL: https://chem.libretexts.org/Bookshelves/Analytical_Chemistry/Supplemental_Modules_%28Analytical_Chemistry%29/Instrumentation_and_Analysis/Capillary_Electrophoresis.

[7] Aivar Loopalu. *Töölauarakendus kapillaarelektroforeesi teostamiseks*. [Kasutatud 01.06.2025]. 2019. URL: https://digikogu.taltech.ee/et/item/63c4f4f0-9dea-41c3-9710-a492f74bc113.

[8] Gert Kivimägi. *Kapillaarelektroforeesi andmetöötluse veebirakendus*. [Kasutatud 01.06.2025]. 2019. URL: https://digikogu.taltech.ee/et/Item/299f18d5-b826-40c1-b273-87981a6d1c4d.

[9] Roman Bondarev ja Marina Ivanova. *Kapillaarelektroforeesi andmetöötluse töölaua- ja veebirakendus*. [Kasutatud 01.06.2025]. 2020. URL: https://digikogu.taltech.ee/et/Item/c3847c61-9ea7-4484-b6ea-f3bf903ab6ba.

[10] Johann Veispak. *Täppisväetamise kasutajaliides*. [Kasutatud 01.06.2025]. 2022. URL: https://digikogu.taltech.ee/et/item/c514b132-2117-4493-a2b5-35c8c5d2979e.

[11] *JHipster*. [Kasutatud 04.06.2025]. URL: https://www.jhipster.tech/.

[12] Murugiah Souppaya ja Karen Scarfone. *Guide to Enterprise Patch Management Planning: Preventive Maintenance for Technology (NIST SP 800-40 Rev. 4)*. [Kasutatud 02.05.2025]. URL: https://doi.org/10.6028/NIST.SP.800-40r4.

[13] Sonar. *Technical debt developer's guide*. [Kasutatud 02.05.2025]. URL: https://www.sonarsource.com/learn/technical-debt/.

[14] *PostgreSQL*. [Kasutatud 04.06.2025]. URL: https://www.postgresql.org/.

[15] *Angular*. [Kasutatud 04.06.2025]. URL: https://angular.dev/.

[16] *Highcharts*. [Kasutatud 04.06.2025]. URL: https://www.highcharts.com/.

[17] *JSON Web Token*. [Kasutatud 04.06.2025]. URL: https://jwt.io/introduction.

[18] *Spring Boot*. [Kasutatud 04.06.2025]. URL: https://spring.io/projects/spring-boot.

[19] *Liquibase*. [Kasutatud 04.06.2025]. URL: https://www.liquibase.com/.

[20] JHipster. *Upgrading an application*. [Kasutatud 04.06.2025]. URL: https://www.jhipster.tech/upgrading-an-application/.

[21] *React Native*. [Kasutatud 03.06.2025]. URL: https://reactnative.dev/.

[22] *Flutter*. [Kasutatud 03.06.2025]. URL: https://flutter.dev/.

[23] *Figma*. [Kasutatud 03.06.2025]. URL: https://www.figma.com/.

[24] Pekka Abrahamsson *et al.* „Agile software development methods: Review and analysis“. *arXiv preprint arXiv:1709.08439* (2017).

[25] *Docker*. [Kasutatud 04.06.2025]. URL: https://www.docker.com/.

[26] *Apache Maven Project*. [Kasutatud 04.06.2025]. URL: https://maven.apache.org/.

[27] *NGINX*. [Kasutatud 04.06.2025]. URL: https://nginx.org/.

[28] *Bootstrap*. [Kasutatud 04.06.2025]. URL: https://getbootstrap.com/.

[29] *Bootswatch*. [Kasutatud 04.06.2025]. URL: https://bootswatch.com/.

[30] *Expo*. [Kasutatud 04.06.2025]. URL: https://expo.dev/.

[31] *JUnit 5*. [Kasutatud 04.06.2025]. URL: https://junit.org/junit5/.

[32] *Spring TestContext Framework*. [Kasutatud 04.06.2025]. URL: https://docs.spring.io/spring-framework/reference/testing/testcontext-framework.html.

[33] *Chrome DevTools*. [Kasutatud 04.06.2025]. URL: https://developer.chrome.com/docs/devtools.

[34] *Jest · Delightful JavaScript Testing*. [Kasutatud 04.06.2025]. URL: https://jestjs.io/.

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Meie, Anna Meier, Mari Piirsalu ja Andreas Ayrton Luhala

1. Anname Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Täppisväetamise veebi- ja mobiilirakendus", mille juhendaja on Evelin Halling
    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. Oleme teadlikud, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autoritele.
3. Kinnitame, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

04.06.2025

[1] Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – Proksi konfiguratsioon

```
upstream api_backend {
    server api_backend:8080;
}

# ── Redirect all HTTP to HTTPS ──────────────────────
server {
    listen 80 default_server;
    server_name electrophoresis.cs.taltech.ee 193.40.244.105;

    # Redirect any request to https
    return 301 https://$host$request_uri;
}

# ── Main HTTPS server ───────────────────────────────
server {
    listen 443 ssl http2 default_server;
    server_name electrophoresis.cs.taltech.ee 193.40.244.105;

    # points to certs inside the container
    ssl_certificate      /etc/nginx/ssl/electrophoresis.cs.taltech.ee.
        crt;
    ssl_certificate_key  /etc/nginx/ssl/electrophoresis.cs.taltech.ee.
        key;

    # security hardening
    ssl_protocols        TLSv1.2 TLSv1.3;
    ssl_ciphers          HIGH:!aNULL:!MD5;
    ssl_session_cache    shared:SSL:10m;
    ssl_session_timeout  10m;
    ssl_prefer_server_ciphers on;

    # ── SPA ─────────────────────────────────────────
    location / {
```

```
        proxy_pass          http://capillaryelectrophoresis-frontend-app
            ;
        proxy_set_header    Host              $host;
        proxy_set_header    X-Real-IP         $remote_addr;
        proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
        expires             1h;
    }

    # -- API / Actuator / Swagger ------------------------------
    location ~ ^/(api|services|management|swagger-resources|v2/api-docs
        |v3/api-docs|h2-console|auth|health)/ {
        proxy_pass          http://api_backend;
        proxy_set_header    Host              $host;
        proxy_set_header    X-Real-IP         $remote_addr;
        proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

Lisa 3 – Veebirakenduse aadress

https://electrophoresis.cs.taltech.ee/